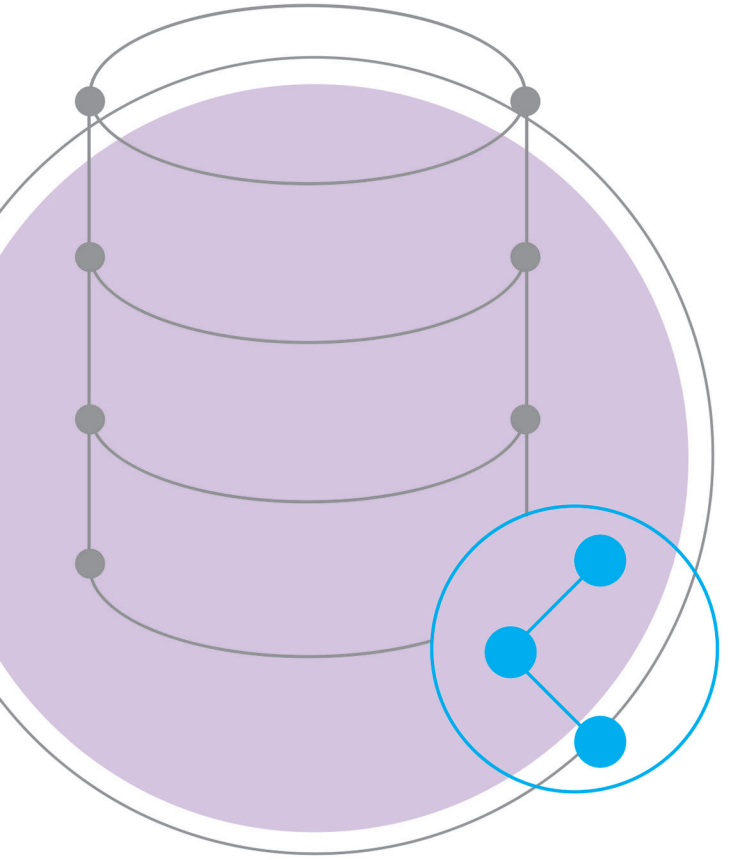
ВИДАВНИЦТВО
РАНОК

Віктор Руденко,
Наталія Речич, Валентина Потієнко

11

ІНФОРМАТИКА

Профільний рівень



Інтернет-
підтримка

Віктор Руденко,
Наталія Речич, Валентина Потієнко

ІНФОРМАТИКА

ПРОФІЛЬНИЙ РІВЕНЬ

ПІДРУЧНИК ДЛЯ 11 КЛАСУ
ЗАКЛАДІВ ЗАГАЛЬНОЇ СЕРЕДНЬОЇ ОСВІТИ

РЕКОМЕНДОВАНО
МІНІСТЕРСТВОМ ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВ
ВИДАВНИЦТВО «РАНОК»
2019

УДК 004:37.016 (075.3)
Р83

Рекомендовано Міністерством освіти і науки України
(наказ Міністерства освіти і науки України від 12.04.2019 № 472)

Видано за рахунок державних коштів. Продаж заборонено

Руденко В. Д.
Р83 Інформатика (профільний рівень) : підруч. для 11 кл. закл. загал. серед. освіти / В. Д. Руденко, Н. В. Речич, В. О. Потієнко. — Харків : Вид-во «Ранок», 2019. — 256 с. : іл.
ISBN 978-617-09-5237-0

УДК 004:37.016 (075.3)

Інтернет-підтримка
Електронні матеріали
до підручника розміщено на сайті
interactive.ranok.com.ua

© Руденко В. Д., Речич Н. В.,
Потієнко В. О., 2019
© ТОВ Видавництво «Ранок», 2019

ISBN 978-617-09-5237-0

Шановні учні та учениці!

В 11 класі завершується вивчення шкільного курсу інформатики. Ви досягли певного рівня інформаційної культури і здатні самостійно оволодівати сучасними інформаційними технологіями. Та інформатика — дуже динамічна наука. Її подальші напрямки й темпи розвитку значною мірою визначатимуться рівнем підготовки людей, які мають ґрунтовні знання в цій галузі.

Цього року ви будете працювати з новими програмними засобами, освоїте розробку найпростішої бази даних навчального призначення в середовищі Access, навчитеся створювати веб-сайти з використанням систем керування вмістом, реалізовувати базові алгоритми засобами мови програмування Python і середовища програмування IDLE та створювати й налаштовувати програми за розробленими алгоритмами, опануєте основні етапи та методологію розробки програмного забезпечення тощо.

Бажаємо вам успіхів, *авторський колектив*

Підручник, який ви тримаєте в руках, — ваш надійний помічник. У ньому ви знайдете завдання для самостійного виконання — виконуйте їх на комп'ютері з натхненням, повторюйте теоретичний матеріал і викладайте основні положення на папері.

Матеріали на підтримку практичних робіт ви знайдете на сайті interactive.ranok.com.ua, де також маєте змогу пройти комп'ютерне тестування з автоматичною перевіркою результату.

Різнорівневі питання для перевірки знань і завдання для самостійного виконання відповідають рівням навчальних досягнень:

 — початковий і середній рівні

 — достатній рівень

 — високий рівень

У тексті також використано позначення:

— питання на повторення

— означення, висновок

— зверніть увагу

— цікаво знати

— завдання для виконання й обговорення в парах або групах

— вправи для домашнього виконання

Розділ 1. БАЗИ ДАНИХ

1. Загальні відомості про бази даних

1.1. Поняття бази даних і системи управління базами даних

Згадайте, з якими базами даних вам доводилося працювати раніше. Наведіть приклади баз даних.

Нині важко уявити роботу банку, системи резервування й продажу авіаквитків, великого супермаркету, готельного комплексу, морського порту тощо без використання інформаційних систем, основою яких є бази даних (БД).

База даних — це сховище організованої сукупності даних різного типу, які відображують стан об'єктів певної предметної галузі та зв'язки між ними.

Предметною галуззю називають сферу застосування конкретної БД, наприклад школа, будівельна організація, аеропорт, банк, поліклініка, супермаркет тощо.

Об'єктом предметної галузі є те, про кого або про що зберігаються дані в БД. Якщо предметною галуззю є, наприклад, школа, то її об'єктами можуть бути учні, вчителі, директор школи, кабінети.

Кожен об'єкт характеризується сукупністю атрибутів, або властивостей (приклад 1). Далі об'єкти БД будемо позначати так: великими літерами — назва об'єкта, у круглих дужках — перелік його атрибутів, які відокремлюються один від одного комою. Наприклад, об'єкт ПОТЯГ можна позначити так: ПОТЯГ (номер потягу, станція відправлення, час відправлення, кінцева станція, час прибуття на кінцеву станцію).

За структурою даних БД поділяють на дві основні групи: документальні й фактографічні (рис. 1.1).

У фактографічних БД кожен атрибут об'єкта має певну сукупність значень, тобто елементів даних, які є найменшими неподільними одиницями даних. Наприклад, атрибут Центр (див. рис. 1.1) має значення Полтава і Хмельницький, атрибут Площа — значення 28 748 і 20 600, а атрибут Районів — значення 25 і 13.

БД є однією з найважливіших складових сучасної інформаційної системи, побудованих на основі комп'ютерних систем і мереж. Робота з БД у таких системах здійснюється за допомогою спеціальної мови БД або програмного забезпечення — системи управління базами даних (СУБД).

Уперше термін *database* (база даних) з'явився на початку 60-х років XX ст. і був уведений у вжиток на симпозіумах, організованих фірмою System Development Corporation (США) у 1964 і 1965 ро+289ках. Широкого розповсюдження в сучасному розумінні цей термін набув у 1970-ті роки з розвитком ЕОМ.

Приклад 1. Об'єкт УЧЕНЬ може мати такі атрибути: прізвище, ім'я, рік народження, домашня адреса, школа, клас, зріст, а об'єкт АВТОМОБІЛЬ — такі: модель, потужність двигуна, максимальна швидкість, вантажопідйомність.

Рис. 1.1. Класифікація баз даних за структурою даних

Система управління базами даних — це інструмент, призначений насамперед для створення структури БД, уведення й оновлення даних, пошуку необхідних даних та їх опрацювання за певним алгоритмом.

Оскільки до БД може звертатися велика кількість користувачів, то важливою функцією СУБД є забезпечення цілісності й безпечності даних. Окрім функцій, безпосередньо пов'язаних зі створенням і підтримкою БД, окремі СУБД виконують також функцію підтримки спеціалізованих мов програмування, що мають загальну назву «мови баз даних». Наприклад, СУБД Access 2016 підтримує мову запитів SQL. Отже, для створення якісних БД і кваліфікованої роботи з ними необхідно добре опанувати СУБД.

СУБД класифікують за багатьма ознаками. До найголовніших можна віднести *призначення*, *модель даних*, *спосіб доступу*. Спрощену схему класифікації СУБД подано на рис. 1.2.

Нині фактичним стандартом мови БД є мова SQL. Однак у деяких випадках доводиться користуватися й іншими мовами програмування, наприклад мовою VBA. Розробники БД засобами СУБД та іншими мовами програмування можуть розробляти прикладні програми, за допомогою яких користувач натисканням однієї кнопки може отримати з БД необхідні дані або опрацювати їх за певним алгоритмом. Наприклад, обчислити суму реалізованих у супермаркеті певних товарів за добу, нарахувати заробітну платню працівникам фірми або отримати інформацію про наявність вільних місць у готелях міста Відня, що не дорожчі ніж 200 євро за добу.

Рис. 1.2. Класифікація систем управління базами даних

Отже, взаємодія користувача з БД може здійснюватися як засобами СУБД, так і за допомогою прикладних програм, що пояснюється схемою (рис. 1.3).

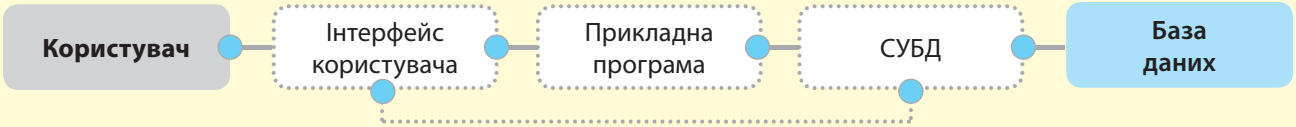


Рис. 1.3. Варіанти взаємодії користувача з базою даних

Приклад 2. Якщо в системі продажу квитків на потяг замовлення на квиток із будь-яких причин не виконано, у БД жодних змін щодо наявності квитків внесено не буде, тобто відбудеться відкат.

Приклад 3. Сутність ХОЛОДИЛЬНИК характеризується такими атрибутами, як назва, маса, ціна, потужність. Конкретний холодильник є екземпляром сутності ХОЛОДИЛЬНИК. Атрибут, що є унікальним, тобто однозначно визначає екземпляр сутності, називають ключем.

Сучасні БД мають величезні обсяги даних і зберігаються в комп'ютерних системах на жорстких магнітних дисках. Користувач позбавлений необхідності знати тонкощі фізичного розміщення даних на них. Ця функція повністю реалізується СУБД разом з операційною системою.

Важливою функцією СУБД є також керування транзакціями. *Транзакція* — це послідовність операцій над даними, яка сприймається СУБД як єдине ціле.

Якщо всі операції з послідовності виконано успішно, то вважається, що й транзакцію завершено успішно. Усі зміни даних, виконані за цією транзакцією, вносяться в зовнішню пам'ять. Та якщо хоча б одну операцію послідовності завершено невдало, транзакція вважається невиконаною і здійснюється відкат — скасування змін у всіх даних, виконаних у процесі транзакції, та повернення БД до початкового стану виконання транзакції (приклад 2).

Ще однією важливою функцією СУБД є так звана *журналізація*, під якою розуміють облік уведених у БД змін. Перед виконанням потрібних змін їх вносять до спеціального журналу. У разі апаратного або програмного збою БД можна повністю відновити за допомогою архівної копії БД і журналу.

Одним із засобів моделювання предметної галузі на етапі проектування БД є модель сутність — зв'язок. Основними поняттями такої моделі є сутність, атрибут і зв'язок.

Сутність — це деякий об'єкт реального світу. Вона має екземпляри, які відрізняються один від одного значеннями атрибутів. Атрибут — це властивість сутності. Зв'язок фактично встановлює взаємодію між сутностями (приклад 3).

У реляційних БД сутності відповідає таблиця, а екземпляру — запис.

Запитання для перевірки знань

1. Що називають предметною областю БД?
2. Наведіть приклади властивостей об'єкта СМАРТФОН.
3. Як позначають об'єкти БД?
4. Як поділяються БД за структурою?
5. Які БД називають фактографічними?
6. Наведіть означення БД.
7. Назвіть основні функції СУБД.
8. Поясніть сутність транзакції.

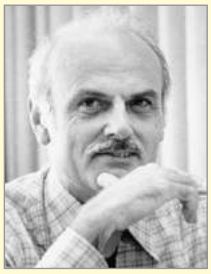
1.2. Поняття моделі даних

Пригадайте означення моделі та моделювання. Що, на вашу думку, означає термін «модель даних»?

Як вже зазначалося, об'єкти предметної галузі характеризуються сукупністю *атрибутів* (*властивостей*) та їх значеннями. Одне значення атрибута називають елементарною одиницею даних. Наприклад, для об'єкта АВТОМОБІЛЬ його елементарними одиницями можуть бути марка — Volkswagen і двигун — дизельний.

Таким чином, об'єкти БД характеризуються сукупністю елементарних одиниць даних, між якими повинні бути встановлені однозначні зв'язки. Це означає, що основою будь-якої структури даних є відображення елементарної одиниці даних у вигляді трійки: <об'єкт, атрибут об'єкта, значення атрибута>, наприклад: <учень, прізвище, Костирко>; <учень, клас, 11>.

Дані, що зберігаються в БД, мають певну логічну структуру, тобто **описуються деякою моделлю даних**, яка підтримується відповідною СУБД.



У 1970-х роках американський математик Е. Кодд розробив теоретичні основи реляційної моделі даних. У 1981 році за вагомий внесок у теорію і практику створення реляційних БД учений отримав премію Тюрінга.

Існують різні способи відображення зв'язків між даними, тобто різні моделі даних. Нині є три класичні моделі даних: *ієрархічна*, *мережева* і *реляційна*. Розвиваються й інші моделі даних, засновані на класичних, наприклад *об'єктно-реляційна*.

Таким чином, модель даних визначає, як відбувається об'єднання даних у структури. Вона також визначає можливі операції над даними й обмеження на їх значення.

Ієрархічна і мережева моделі засновані на таких поняттях, як *рівень*, *вузол*, *зв'язок*. Приклад структури і стислий опис сутності цих моделей подано на рис. 1.4.

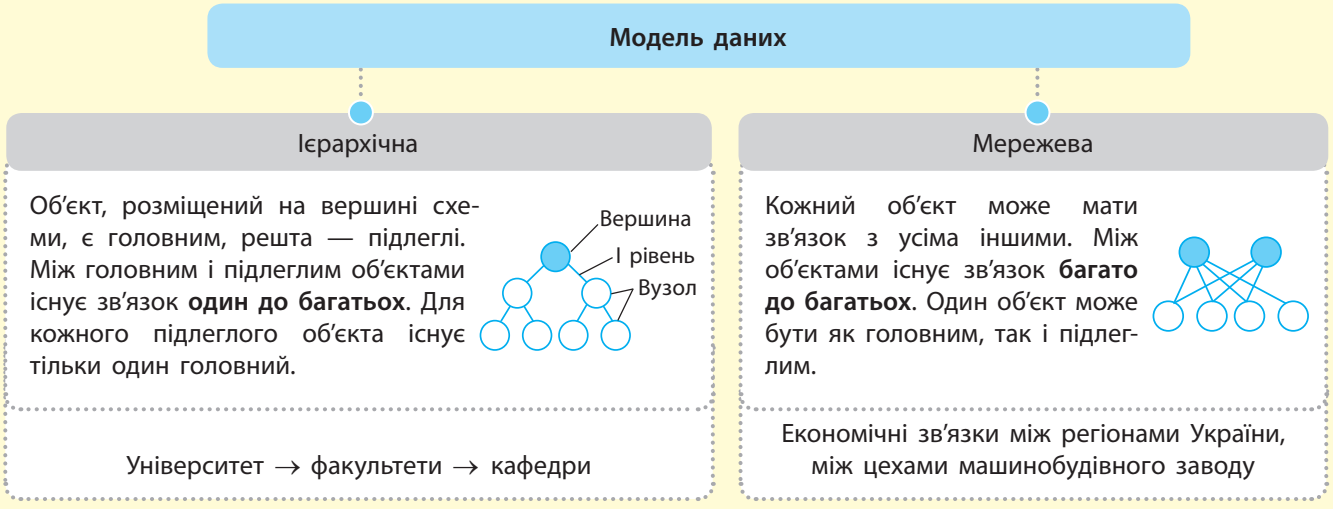


Рис. 1.4. Структури ієрархічної і мережевої моделей даних

Із рис. 1.4 видно, що в цьому випадку *ієрархічна* модель містить три рівні об'єктів. На верхньому рівні міститься головний об'єкт, на другому рівні розташовано два вузли, на третьому — три вузли. Зв'язки між вузлами зображено стрілками. Як бачимо, вузли верхнього рівня мають зв'язки з вузлами найближчого нижнього рівня. *Мережева* модель містить два рівні (їх може бути скільки завгодно), на кожному з яких є два вузли. Звернемо увагу на те, що в цій моделі кожний вузол може мати зв'язок із будь-яким іншим вузлом.

Основним недоліком ієрархічних і мережевих БД є складність їх розроблення, тому нині поширення набула реляційна модель даних — фактографічна база даних, що є набором взаємопов'язаних таблиць. Основна перевага цієї моделі полягає у простоті розроблення БД і систем управління ними.

Найпростіша БД містить одну таблицю, а складні — десятки й навіть сотні таблиць. Розглянемо приклад найпростішої реляційної БД, яка містить тільки одну таблицю УЧНІ (табл. 1.1).

У реляційних моделях об'єкти і взаємозв'язки між даними подаються за допомогою відношень. Порядок розміщення рядків і стовпців у таблиці є довільним. Таблиці в теорії БД називають **відношеннями**, рядки — **записами**, а стовпці — **полями**.

Таблиця 1.1. УЧНІ

Номер	Прізвище	Дата народження	Адреса	Клас	Зріст, см
1	Колот А. І.	07.02.2002	Зоряна, 2, кв. 7	10	172
2	Таранов С. О.	02.06.2003	Поштова, 3, кв. 9	9	174
3	Федорчук Ю. А.	30.05.2003	Лісова, 5	9	165

Не кожна таблиця може бути об'єктом БД. Для того щоб таблиця стала об'єктом БД, потрібно виконати її нормалізацію. Сутність нормалізації полягає в тому, що таблиця повинна бути перетворена відповідно до основних вимог.

Основні вимоги до таблиці як об'єкта БД такі:

- кожне поле повинно мати унікальне ім'я;
- усі поля мають бути однорідними, тобто значення елементів одного поля можуть бути лише одного типу (наприклад, тільки числовими, тільки рядковими);
- у таблиці не може бути однакових записів, вони мають відрізнятися значеннями хоча б одного поля;
- таблиця повинна мати ключове поле, або ключ.

Зазвичай таблиця має унікальне поле або кілька полів, які ідентифікують записи. Таке поле називають ключовим (ключем). Воно використовується для швидкого пошуку даних, а також для зв'язування даних із різних таблиць.

Ключ, який містить тільки одне поле, називають простим, а який містить кілька полів — складним. У таблиці УЧНІ складним ключем можна вважати поля Прізвище і Дата народження, оскільки вони однозначно ідентифікують записи.

У таблиці може бути кілька ключів, але тільки один із них можна визнати як первинний. Найкраще первинним ключем вибрати простий ключ і бажано, щоб він мав цілочисловий тип. У цьому випадку операції опрацювання даних виконуватимуться швидше. У таблиці УЧНІ простим є поле з іменем Номер.

У 1973 році американський вчений Чарльз Бахман, один із засновників мережевої моделі БД, отримав премію Тюрінга за керування роботою Data Base Task Group (робоча група по базах даних, США), яка розробила стандартну мову опису даних і маніпулювання даними.

У таблиці часто використовують поле — воно називається лічильником, яке використовується для того, щоб зробити кожний запис унікальним. Крім того, лічильник забезпечує нумерацію записів. У таблиці УЧНІ лічильником є поле з іменем Номер.

Важливо усвідомити, що на основі однієї таблиці можна створити БД будь-якої складності. Таблиця може містити сотні полів і тисячі записів, і працювати з нею досить складно. Щоб не сталося значного дублювання даних, для кожного об'єкта розробляється власна таблиця. А щоб можна було одночасно отримувати дані з кількох таблиць, потрібно встановлювати зв'язки між ними (приклад 1).

В основній таблиці вибирають *первинний* ключ, а в допоміжній — *зовнішній* ключ. Зовнішній ключ повинен однозначно визначати поле основної таблиці. У ньому не може бути даних, відсутніх у первинному ключі, інакше зв'язок буде некоректним. Часто для забезпечення зв'язку між таблицями в допоміжну таблицю спеціально вводять поле з таким самим іменем, що й ім'я первинного ключа основної таблиці. У такому випадку деякі СУБД автоматично встановлюють зв'язок між таблицями. Якщо імена зазначених полів різні, то користувач повинен сам встановити зв'язок між ними. Пояснимо сутність зв'язків між двома таблицями на прикладі 2.

Приклад 1. Для БД фірми в одній таблиці можуть зберігатися дані про співробітників, у другій — дані про їхню заробітну платню, у третій — відомості про постачальників продукції. Такий підхід спрощує подальшу модифікацію БД.

Зв'язки можуть встановлюватися між двома, трьома й більшою кількістю таблиць. Для встановлення зв'язків між двома таблицями одну з них вибирають **основною** (**батьківською**), а іншу — **допоміжною** (дочірньою).

Приклад 2. Нехай у БД будівельної компанії є дві таблиці: табл. 1.2 ПОСТАЧАЛЬНИКИ і табл. 1.3 ТОВАРИ.

Таблиця 1.2. ПОСТАЧАЛЬНИКИ

Фірма	Директор	Телефон
РПЗ-1	Сопко А. І.	345-23-51
БУТ-5	Маслов В. М.	295-44-87
ДМК-2	Бондаренко К. О.	454-98-56

Таблиця 1.3. ТОВАРИ

Матеріал	Маса, кг	Фірма
Бетон	100	РПЗ-1
Бетон	120	БУТ-5
Бетон	200	ДМК-2
Цемент	50	БУТ-5

У табл. 1.2 ПОСТАЧАЛЬНИКИ первинним ключем є поле з іменем Фірма. У табл. 1.3 ТОВАРИ поле з цим іменем не може бути первинним ключем, оскільки в ньому повторюються назви фірм. Воно може бути зовнішнім ключем, тому що його значення збігаються зі значеннями однойменного поля табл. 1.2 ПОСТАЧАЛЬНИКИ. Більше того, вони мають однакове ім'я. За даними цього поля можна встановити зв'язок між двома таблицями.

Щоб дізнатися телефон і прізвище директора фірми, яка постачає 120 тонн бетону і 50 тонн цементу, із табл. 1.3 ТОВАРИ слід вибрати назву фірми БУТ-5 і за її назвою у табл. 1.2 ПОСТАЧАЛЬНИКИ знайти прізвище Маслов В. М., телефон 295-44-87.

Таким чином, зв'язки між таблицями дозволяють отримати дані з кількох таблиць. Окрім того, вони забезпечують цілісність даних у пов'язаних таблицях, якщо з деяких причин сталися зміни в одній таблиці.

Пояснимо сутність цілісності даних на прикладі 3 вже розглянутих таблиць.

Приклад 3. Припустимо, що зв'язок між табл. 1.2 ПОСТАЧАЛЬНИКИ і табл. 1.3 ТОВАРИ не встановлено. Із табл. 1.2 випадковим чином вилучено запис про те, що директором фірми є Маслов В. М., а в табл. 1.3 всі дані збереглися, тобто є цілісними. Ця ситуація відображена у табл. 1.4 ПОСТАЧАЛЬНИКИ.

Тепер прізвище директора фірми БУТ-5 і його телефон невідомі. Вважається, що у цьому випадку трапилося порушення цілісності даних, і ситуація має бути автоматично виявлена.

Таблиця 1.4. ПОСТАЧАЛЬНИКИ

Фірма	Директор	Телефон
РПЗ-1	Сопко А. І.	345-23-51
ДМК-2	Бондаренко К. О.	454-98-56

Між таблицями можуть існувати 4 види зв'язку (рис. 1.5).

Рис.1.5. Види зв'язку між двома таблицями

Найчастіше між таблицями реляційної БД існує зв'язок один-до-багатьох.

Запитання для перевірки знань

1. Що називають елементарною одиницею даних у БД?
2. Назвіть основні моделі даних у БД.
3. Поясніть сутність ієрархічної моделі даних.
4. Які існують види зв'язку між таблицями?
5. Поясніть сутність реляційної моделі даних.
6. Які поля таблиць називають ключем?
7. Які існують ключі в таблицях БД?
8. Наведіть означення моделі даних.
9. Назвіть основні вимоги до таблиць БД.
10. У чому полягає сутність забезпечення цілісності даних БД?

1.3. Основні відомості про систему управління базами даних Access

СУБД призначені для створення й супроводу БД. Спробуйте конкретизувати їхні основні функції.

Історія розвитку БД і систем управління ними налічує кілька етапів. За цей час розроблено багато СУБД, наприклад dBase, FoxPro, Oracle 8.4, MS SQL Server 7.0, SQL Base, MS Access 7 та ін. Усі вони по-різному працюють із об'єктами і мають різні функціональні можливості. Та попри це більшість із них спирається на єдиний комплекс основних понять, що дає нам можливість розглянути одну систему та узагальнити її поняття, прийоми й методи на весь клас СУБД. Далі розглядатимемо одну з найпоширеніших сьогодні СУБД — Access 2016.

СУБД Access 2016 входить до складу пакета Microsoft Office і призначена для створення й підтримки роботи з реляційними БД. Розглянемо її основні об'єкти та їх призначення (рис. 1.6).

СУБД Access 2016 функціонує під керуванням ОС Windows. Системні вимоги: бажано, щоб процесор мав частоту не менше за 800 МГц, обсяг оперативної пам'яті — не менше за 512 Мб, вільний обсяг пам'яті на жорсткому диску — не менше за 2 Гб.

Рис.1.6. Об'єкти системи управління базами даних та їх призначення

Запуск системи Access 2016 можна здійснити стандартними способами, що передбачені в ОС Windows.

Після запуску системи на екрані монітора з'явиться її стартове вікно (рис. 1.7).

На ділянці вікна зліва відображено імена БД, з якими користувач працював останнім часом, на ділянці справа — піктограми шаблонів і піктограма порожньої БД.

У середовищі Access 2016 БД можна створити «з нуля», тобто повністю самостійно, або скористатися шаблонами, які має система. Якщо наявних шаблонів не вистачає, їх можна знайти в Інтернеті, скориставшись полем пошуку. Для якісного оволодіння способами створення й супроводу БД

Рис. 1.7. Стартове вікно Access 2016

користувачу-початківцю доцільно спочатку навчитися створювати нову БД.

Серед шаблонів в Access є Пуста база даних, яка слугує для створення нової БД. У подальшому ми будемо використовувати саме цей спосіб.

Розглянемо порядок дій для створення нової БД.

Крок 1

Клацнути піктограму **Пуста база даних**.
Відкриється вікно, зображене на рис. 1.8.

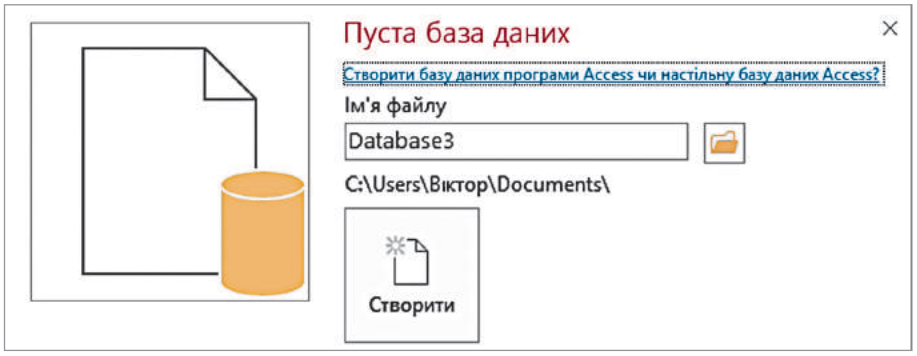


Рис. 1.8. Вікно для створення нової бази даних

Крок 2

У рядок **Ім'я файлу** ввести ім'я файла майбутньої БД, наприклад atb, натиснути кнопку **Знайти розташування для бази даних**, що розташована праворуч від цього рядка.

Крок 3

У вікні **Створення бази даних**, що відкриється, вибрати місце збереження файла БД, наприклад диск F:, і натиснути кнопку **OK**, а потім — кнопку **Створити**.

У результаті цих дій файл БД буде зареєстровано в кореневому каталозі диска F:, а на екрані з'явиться вікно для створення таблиці 1 (рис. 1.9).

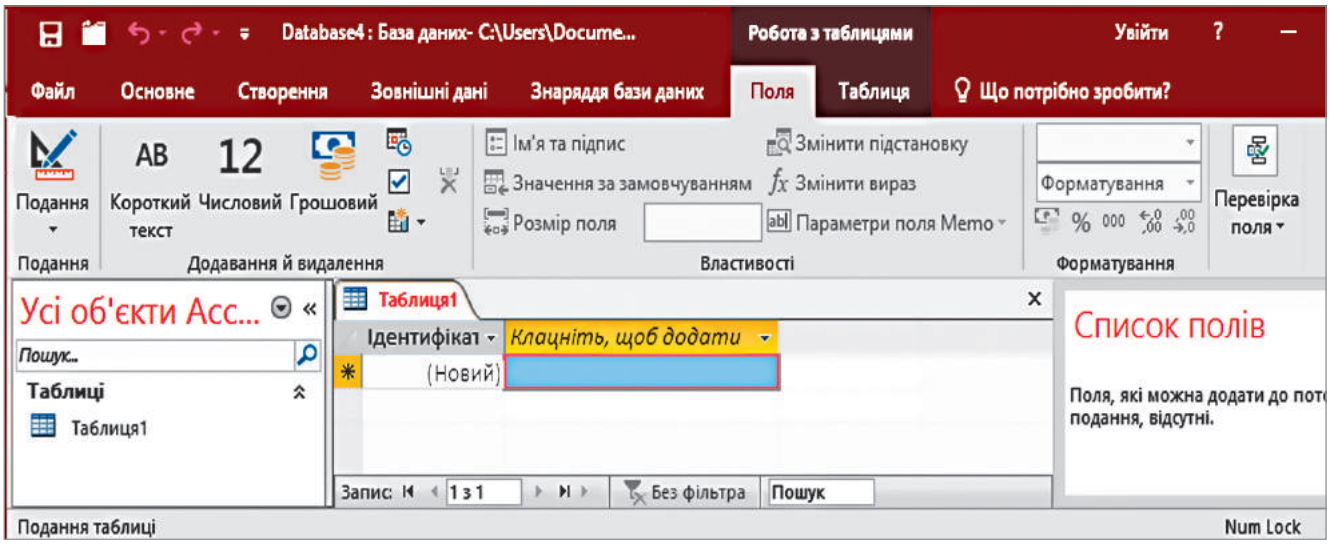


Рис. 1.9. Вікно для створення таблиці

Основним об'єктом вікна є горизонтальна стрічка, на якій розташовано команди й інструменти Access 2016. Їх призначення будемо розглядати поступово під час безпосереднього використання. Розглянемо ті, які потрібні на цьому етапі.

У верхній частині стрічки розміщено вкладки Файл, Основне, Створення та ін. Зміст команд та елементів керування, які відображені на стрічці, залежить від того, яку на цей момент вкладку відкрито.

На рис. 1.9 відкрито вкладку Поля, і в цьому випадку всі команди й елементи керування згруповані у розділи Подання, Додавання й видалення, Форматування. Якщо відкрити іншу вкладку, наприклад Основне, з'являться нові елементи й команди, які буде згруповано в нові розділи.

Стисло ознайомимося з призначенням вкладок вікна.

Вкладка Основне містить команди й елементи керування, які найчастіше використовуються в процесі роботи з БД. Зокрема тут містяться команди для роботи з буфером обміну, форматування тексту, сортування й фільтрування даних та ін.

Вкладка Створення містить команди для створення таблиць, запитів, форм та інших об'єктів БД, вкладка Зовнішні дані — команди для експортування й імпортування даних, вкладка Знаряддя бази даних — команди встановлення зв'язків між таблицями, аналізування й переміщення даних між програмами та ін.

У лівій частині екрана вміщено Усі об'єкти Access — панель переходів, на якій можуть відображатися назви всіх створених об'єктів, між якими можна здійснювати перехід простим натисненням відповідних назв. Праворуч від панелі переходів міститься область редагування, у якій можуть одночасно відображатися таблиці, запити та інші об'єкти БД.

Відкрити вже створену БД можна за допомогою кнопки Відкрити на панелі швидкого доступу, а щоб закрити, потрібно скористатися командою Закрити, що розташована на вкладці Файл.

У кожний момент часу Access 2016 підтримує роботу тільки з однією БД. Число користувачів, які одночасно працюють із БД, може досягати 255. Імена об'єктів можуть включати до 64 символів, максимальний обсяг файла БД становить 2 Гб.

Щоб згорнути й розгорнути стрічку, слід скористатися відповідними командами кнопки **Настроїти панель швидкого доступу**, яка міститься на панелі швидкого доступу, або натиснути праву кнопку миші в будь-якому місці та виконати команду **Згорнути стрічку** в контекстному меню, що з'явиться.

Запитання для перевірки знань

1. Назвіть основні об'єкти Access 2016 та їх призначення.
2. Що розміщено в області **Усі об'єкти** Access?
3. Як можна відкрити вже створену БД?
4. Поясніть сутність створення БД «з нуля».
5. Які основні дії можна виконувати на вкладці **Створення**?
6. Які основні системні вимоги для Access 2016?

Завдання для самостійного виконання

1. Запустіть систему Access 2016. Створіть на жорсткому диску файл БД з іменем **Mybasa**. Переконайтеся, що файл зареєстровано.
2. Проаналізуйте призначення об'єктів початкового вікна БД, відкриваючи різні вкладки.
3. Відкрийте та проаналізуйте призначення команд кнопки **Настроїти панель швидкого доступу.**
4. Відкрийте і проаналізуйте вміст вкладки **Основне**.

2. Таблиці

Створюючи БД Access, користувачі зберігають дані в таблицях. Об'єкти БД залежать від структури таблиць, тому розробку БД потрібно починати зі створення власне таблиць і лише після цього можна переходити до будь-яких інших об'єктів.

2.1. Створення й уведення структури таблиць

Поміркуйте, з якою метою застосовуються таблиці в БД. Якою є основна функція таблиць?

У середовищі Access 2016 існують такі інструменти створення таблиць, як **Конструктор таблиць**, **Майстер таблиць**, **Режим таблиць**.

Таблиці в середовищі Access 2016 можна створювати різними способами. Та найчастіше їх створюють розробники БД — починаючи від проектування на папері й закінчуючи створенням електронних працездатних таблиць. Саме такий варіант ми розглянемо далі.

Розробці таблиць передує детальний аналіз предметної області, визначення вимог, змісту та структури документів, які необхідно отримати. У процесі розроблення слід визначити кількість таблиць, а також назву, структуру і вміст кожної таблиці. Далі порядок розроблення й опрацювання таблиць розглянемо на прикладі предметної галузі Магазини.

Приклад 1. Припустимо, що БД деякої мережі магазинів містить дві таблиці: табл. 2.1 МАГАЗИНИ і табл. 2.2 КАДРИ. Визначимо їх структуру і вміст.

Таблиця 2.1. МАГАЗИНИ

Номер магазину	Адреса	Директор	Телефон	Працівників
21	вул. Паркова, 33	Коцюба П. М.	234-54-63	20
31	вул. Печерська, 21	Борзов А. С.	234-22-98	13
6	вул. Річкова, 24	Середа К. М.	234-67-92	15

Таблиця 2.2. КАДРИ

Номер справи	Прізвище	Посада	Рік народження	Освіта	Стаж	Оклад	Номер магазину
105	Сокіл Т. Л.	Касир	1960	Середня	27	3500	6
132	Таран В. Д.	Диспетчер	1973	Вища	15	4000	31
120	Рябко Р. П.	Експерт	1981	Вища	8	4200	21
111	Семко М. М.	Диспетчер	1970	Середня	16	4000	21
115	Горошко Ф. Р.	Диспетчер	1975	Середня спеціальна	17	4000	31
116	Раков Г. П.	Аналітик	1965	Вища	19	4500	21
109	Шрамко Т. Л.	Диспетчер	1961	Середня	24	4000	6

Головною таблицею вважатимемо табл. 2.1 МАГАЗИНИ, а допоміжною — табл. 2.2 КАДРИ.

Найпотужнішим і універсальним інструментом створення таблиць є Конструктор таблиць. Далі розглянемо цей спосіб, інші способи простіші, ними можна оволодіти самостійно.

Після визначення структури таблиць слід визначити типи полів з урахуванням тих типів, із якими може працювати Access. У середовищі Access 2016 використовуються типи даних, перелік яких наведено на рис. 2.1.

Кожен із наведених типів даних має власний набір властивостей. Деякі властивості є унікальними, тобто містяться тільки в одному конкретному типі даних, а деякі є загальними, тобто містяться в різних типах даних. Далі будемо використовувати в основному такі типи даних: Короткий текст, Довгий текст, Число, Дата й час.

Тип Короткий текст — це послідовність символів завдовжки від 0 до 255, а тип Довгий текст — послідовність символів до 65 536. У полі Число зберігаються числа, їх розмір і конкретний тип визначаються значенням властивості Розмір поля. У полях типу Дата й час зберігаються дата й час різних форматів.

Рис. 2.1. Типи даних Access 2016

Приклад 2. Для більш зручної роботи під час уведення структури табл. 2.1 і табл. 2.2 доцільно створити на папері додаткові таблиці (табл. 2.3 і табл. 2.4) з іменами полів, зазначенням типу даних та описом.

Таблиця 2.3. Структура таблиці МАГАЗИНИ

Ім'я поля	Тип даних	Опис	Властивості
Номер магазину	Число	Первинний ключ	
Адреса	Короткий текст		50
Директор	Короткий текст		40
Телефон	Короткий текст		20
Працівників	Число	Станом на 1 липня	

Таблиця 2.4. Структура таблиці КАДРИ

Ім'я поля	Тип даних	Опис	Властивості
Справа	Число	Первинний ключ	
Прізвище	Короткий текст		40
Посада	Короткий текст		30
Рік народження	Число		
Освіта	Короткий текст		30
Стаж	Число	Станом на 1 січня	
Оклад	Число		
Номер магазину	Число		

На цьому підготовку таблиць для введення в комп'ютер завершено. Перейдемо безпосередньо до створення структур таблиць у режимі конструктора і розглянемо приклад 3.

Приклад 3.

1. Запустимо систему Access 2016 і відкриємо вже створену БД з іменем atb. Для цього в стартовому вікні системи в області Останні натиснемо кнопку миші на імені atb.
2. У вікні, що відкриється, активуємо вкладку Створення. Відкриється вікно зі стрічкою (рис. 2.2).
3. На стрічці натиснемо кнопку Конструктор таблиць. У результаті до БД додасться порожня таблиця (рис. 2.3).

Зверніть увагу на те, що вміст стрічки змінився, і тепер на екрані відкрито вкладку Конструктор (рис. 2.4).

4. Введемо ім'я поля в порожню таблицю (див. рис. 2.3) стандартним способом.

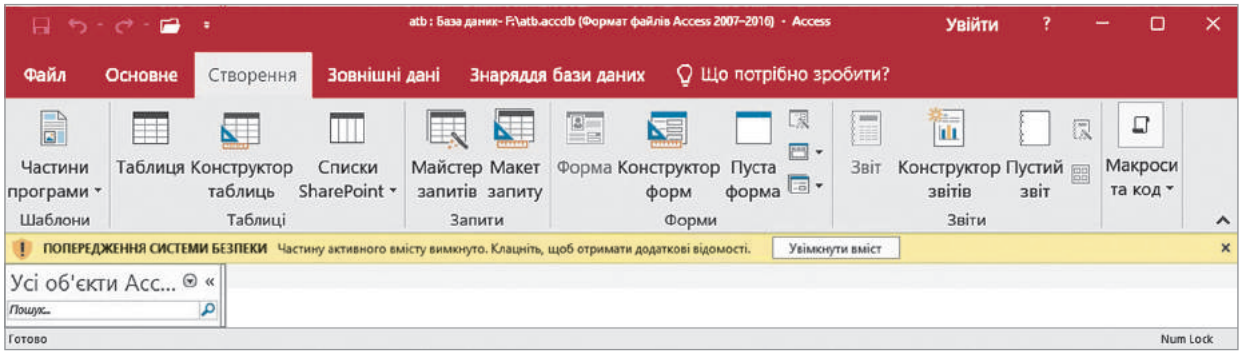


Рис. 2.2. Вікно Access з активованою вкладкою **Створення**

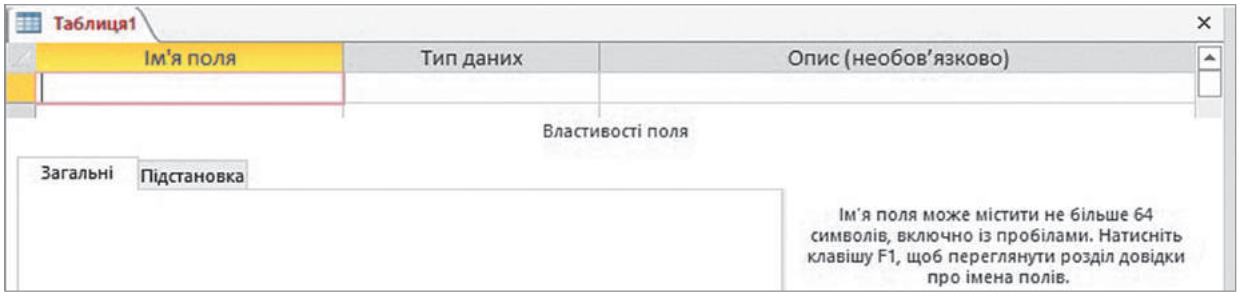


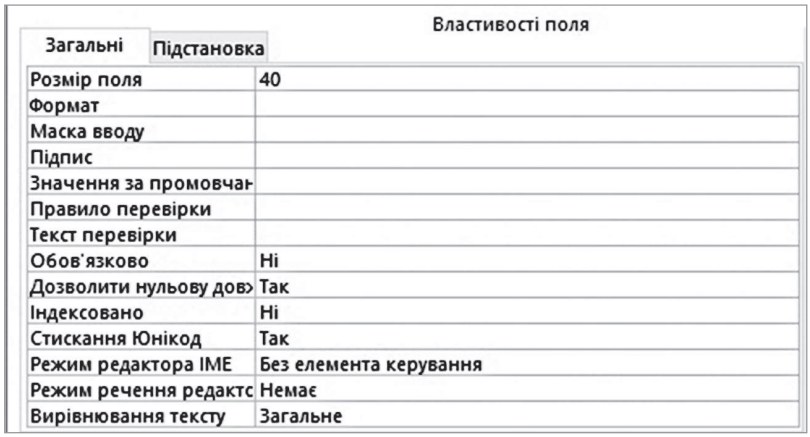
Рис. 2.3. Порожня таблиця в режимі конструктора

Рис. 2.4. Стрічка з активованою вкладкою **Конструктор**

5. Введемо типи полів. Їх краще не вводити з клавіатури, а вибрати зі списку типів (див. рис. 2.1). Щоб відкрити цей список, необхідно у певній клітинці поля Тип даних клацнути кнопку Прапорець, вибрати потрібний тип даних і встановити його властивість. Перелік властивостей наведено нижче від імен полів (рис. 2.5).

Описувати поля не обов'язково. Опис потрібен для уточнення призначення поля та його допустимих значень.

6. Уведемо у відкриту на екрані порожню таблицю (див. рис. 2.3) дані з табл. 2.3 Структура таблиці МАГАЗИНИ. Вміст таблиці набуде вигляду, як наведено на рис. 2.6.


Властивості поля

Загальні | Підстановка

Розмір поля	40
Формат	
Маска вводу	
Підпис	
Значення за промовчан	
Правило перевірки	
Текст перевірки	
Обов'язково	Ні
Дозволити нульову дов	Так
Індексовано	Ні
Стискання Юнікод	Так
Режим редактора IME	Без елемента керування
Режим речення редактс	Немає
Вирівнювання тексту	Загальне

Рис. 2.5. Властивості полів

МАГАЗИНИ

Ім'я поля	Тип даних	Опис (необов'язково)
Номер магазину	Число	Первинний ключ
Адреса	Короткий текст	
Директор	Короткий текст	
Телефон	Короткий текст	
Працівників	Число	станом на 1 липня

Рис. 2.6. Структура таблиці в режимі конструктора

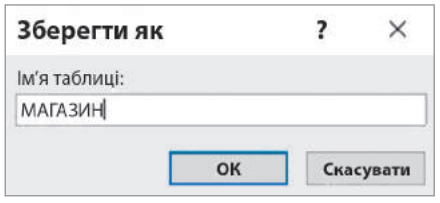


Рис. 2.7. Віконце для збереження таблиці

7. Збережемо таблицю з іменем МАГАЗИНИ. Для цього на панелі швидкого доступу натиснемо кнопку Зберегти (або скористаємося сполученням клавіш Ctrl + S) — відкриється віконце (рис. 2.7). Уведемо в нього ім'я МАГАЗИНИ і клацнемо кнопку OK.

На екран буде виведено попередження (рис. 2.8).

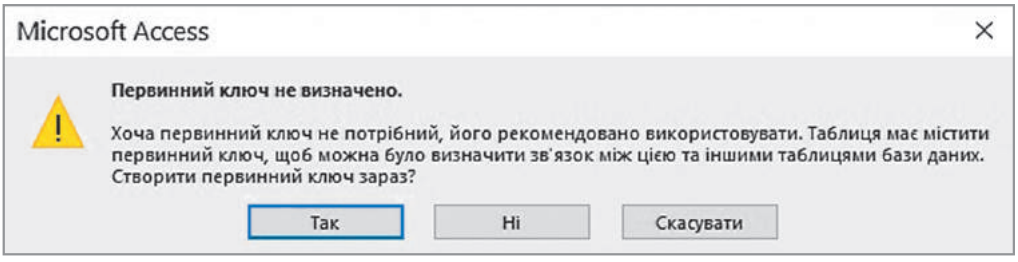


Рис. 2.8. Вікно попередження про невизначений первинний ключ

8. Клацнувши кнопку Так, ми зумовимо створення первинного ключа типу Лічильник. Але на цьому кроці встановлювати первинний ключ не обов'язково, тому клацнемо кнопку Ні. Таблицю буде збережено, а її ім'я з'явиться в області переходів.

9. Закриємо таблицю МАГАЗИНИ, для чого відкриємо її контекстне меню і виконаємо команду Закрити.

10. Аналогічно введемо і збережемо структуру таблиці КАДРИ. Після цього закриємо таблицю.

Зазначимо, що властивості таблиць установлюються у вікні Аркуш властивостей, яке для відкритої таблиці доступне на вкладці Конструктор. У цьому вікні містяться десятки назв властивостей таблиць. Аналогічне вікно властивостей мають і інші об'єкти БД (запити, форми, звіти). Та на цьому етапі вивчення БД ми будемо використовувати властивості, пропоновані за замовчуванням.

Таблицю можна модифікувати на будь-якому кроці створення БД. Але краще це робити до встановлення зв'язків між таблицями і введення в них даних.

Запитання для перевірки знань

1. Опишіть способи створення таблиць.
2. Чому таблиці бажано спочатку створювати на папері?
3. Які дії слід виконати для збереження структури таблиці?
4. Що розуміють під терміном «структура таблиці»?
5. Чому для створення таблиці найчастіше користуються конструктором таблиць?
6. Назвіть основні типи даних таблиць Access.
7. Поясніть порядок уведення структури таблиць Access.
8. Поясніть порядок уведення типу даних поля та його властивостей.

Завдання для самостійного виконання

1. Запустіть СУБД Access 2016 і створіть БД skola з таблицями КЛАСИ (табл. 2.5) й УЧНІ (табл. 2.6).
2. Розробіть на папері структуру табл. 2.5 КЛАСИ і табл. 2.6 УЧНІ. Типи даних полів і їхні властивості виберіть самостійно.
3. Уведіть і збережіть структуру табл. 2.5 КЛАСИ і табл. 2.6 УЧНІ.

Зі створеними БД ви працюватимете протягом періоду їх вивчення, тому працюйте акуратно, не забувайте зберігати дані, не порушуйте структуру таблиць.

Таблиця 2.5. КЛАСИ

Клас	Учнів	Класний керівник	Займаються спортом
9	27	Зотов А. М.	5
10	24	Дерев'янко Н. С.	11
11	25	Терещенко Б. В.	14

Таблиця 2.6. УЧНІ

Прізвище	Адреса	Дата народження	Зріст	Улюблений предмет	Інформатика	Історія	Клас
Ларін Л. К.	Лугова, 5	5.06.2002	163	Математика	11	8	9
Лобов С. П.	Сонячна, 7	9.08.2002	168	Географія	9	9	9
Костенко В. С.	Річкова, 4	13.09.2001	170	Фізика	10	10	10
Рамко Б. В.	Лугова, 9	15.11.2001	169	Інформатика	11	8	10
Пека П. О.	Сонячна, 7	20.12.2001	167	Інформатика	12	9	10
Сергіна В. В.	Лісна, 5	3.03.2002	171	Історія	8	11	10
Хижа Р. А.	Лугова, 2	4.04.2002	172	Історія	9	12	11
Собко О. К.	Вишнева, 5	9.10.2001	175	Інформатика	11	8	11
Настін К. Б.	Лісна, 10	6.11.2001	171	Географія	10	10	11

2.2. Ключові поля, індекси, зв'язування таблиць

Одне з полів таблиці може мати унікальні значення. Для чого, на вашу думку, може використовуватися таке поле?

Згадаємо, що кожна таблиця повинна мати ключове поле (ключ) — поле, значення якого не повторюється в жодному іншому записі. Таблиця може мати кілька ключових полів, але використовується тільки одне з них, яке називають первинним ключем.

Найчастіше первинний ключ складається з одного поля, а як первинний ключ використовується поле типу Лічильник. Якщо в ролі первинного ключа використовуються два і більше полів, його називають складним. Наприклад, у таблиці КАДРИ поле Прізвище не може бути первинним ключем, тому що в мережі магазинів може бути працівник із таким самим прізвищем. А поля Прізвище і Рік народження разом можна вважати таким ключем, оскільки вони, ймовірно, не дублюються.

Ключі у БД відіграють важливу роль — за їх допомогою СУБД ідентифікує об'єкти.

Для створення первинного ключа потрібно відкрити таблицю в режимі конструктора, виділити поле, що використовується як первинний ключ, і натиснути кнопку Ключове поле, що знаходиться в розділі Знаряддя вкладки Конструктор.

Складні ключі бажано не використовувати як первинний ключ, оскільки в цьому випадку ускладнюється процес роботи з БД.

Створити первинний ключ можна також за допомогою контекстного меню певного поля, у якому необхідно виконати команду Ключове поле. Для цього слід відкрити таблицю в режимі конструктора. Скористаємося цим способом, і в таблиці МАГАЗИНИ визначимо як первинний ключ поле Номер магазину. Поряд із назвою цього поля з'явиться зображення ключа (рис. 2.9). Далі збережемо таблицю.

МАГАЗИНИ

Ім'я поля	Тип даних	Опис (необов'язково)
Номер магазину	Число	Первинний ключ
Адреса	Короткий текст	
Директор	Короткий текст	
Телефон	Короткий текст	
Працівників	Число	станом на 1 липня

Рис. 2.9. Структура таблиці МАГАЗИНИ з ключовим полем

Якщо деяке поле в процесі створення структури таблиці оголошено типу Автонумерація (див. рис. 2.1), тобто типу Лічильник, то воно стає ключовим за замовчуванням. Його можна також додати в таблицю навіть у тому випадку, якщо явної потреби в цьому немає.

Поле типу Лічильник обов'язково встановлюється в тому разі, якщо ключ у таблиці взагалі визначити неможливо.

Розглянемо тепер сутність і порядок індексування таблиць.

Індексування — це процес створення додаткових таблиць для певного поля. Ці таблиці (їх ще називають простими індексними таблицями) зазвичай містять лише одне поле, у якому зберігаються вказівники на певні записи таблиці. За допомогою вказівників визначають порядок розміщення записів, упорядкованих за значенням цього поля (приклад 1).

Приклад 1. Індексна таблиця для поля Директор таблиці МАГАЗИНИ матиме такі значення:

2
1
3

Цифра 2 в першому рядку означає, що першим за алфавітом у таблиці є друге прізвище (Борзов А. С.), цифра 1 — що другим за алфавітом є перше прізвище (Коцюба П. М.), а цифра 3 — що третім за алфавітом є третє прізвище (Середа К. М.).

Індексних таблиць для певної таблиці БД може бути кілька, наприклад за кількістю працівників, за номерами магазинів тощо.

Головним призначенням індексних таблиць є підвищення швидкості пошуку необхідних даних (інколи вона зростає до 5 разів). Щоб знайти деякий запис у таблиці, в Access спочатку потрібно знайти його положення в індексі, потім вибрати з нього місце запису в таблиці, що використовується для пошуку даних.

Поля, значення яких змінюються часто, індексувати недоцільно. Для однієї таблиці бажано мати не більш ніж 5 або 6 індексних таблиць.

Зазначимо, що первинний ключ завжди індексований. За замовчуванням записи таблиці виводяться відсортованими за його значеннями. У процесі введення даних у таблицю обов'язково перевіряється значення первинного ключа на дублювання. Якщо значення дублюється, введення запису блокується. Значення первинного ключа типу Лічильник у процесі введення даних формується автоматично (приклад 2).

Приклад 2. Створимо просту індексну таблицю для полів Директор і Працівників таблиці МАГАЗИНИ. Для цього відкриємо таблицю в режимі конструктора, виберемо поле Директор і в розділі Властивості поля в рядку Індексовано ввімкнемо перемикач Так (Без повторень), оскільки малоймовірно, що в цій мережі магазинів буде два директори з однаковими прізвищами. Для поля Працівників увімкнемо перемикач Так (Повторення дозволені), тому що в магазинах може бути однакова кількість працівників.

Основна вимога до ключів така: значення зовнішнього ключа мають збігатися зі значеннями первинного ключа головної таблиці. Імена цих ключів можуть бути різними, але якщо імена однакові, то процес зв'язування таблиць буде простішим.

Під час зв'язування двох таблиць одна з них вважається головною, а інша — допоміжною. Первинний ключ головної таблиці зв'язується із зовнішнім ключем допоміжної.

Часто первинний ключ таблиці штучно вводять у другу таблицю саме з метою їх зв'язування. Цей ключ не є первинним ключем другої таблиці, тому що його значення можуть повторюватися. Наприклад, поле Магазин є первинним ключем таблиці МАГАЗИНИ, а в таблиці КАДРИ це поле є зовнішнім ключем, тому що в ній значення цього поля дублюються.

Приклад 3. Розглянемо порядок створення зв'язку на прикладі таблиць МАГАЗИНИ і КАДРИ (табл. 2.1 і 2.2 відповідно).

1. Завантажимо БД atb й у вікні, що відкриється, активуємо вкладку Знаряддя бази даних. Далі натиснемо кнопку Зв'язки. Відкриється вікно Відображення таблиці (рис. 2.10).
2. У вікні Відображення таблиці виберемо таблиці, які потрібно зв'язати (у нашому випадку обидві таблиці), і натиснемо кнопку Додати. На екрані з'являться ці таблиці з іменами їх полів.
3. Установимо курсор на первинному ключі таблиці МАГАЗИНИ, натиснемо кнопку миші і, не відпускаючи її, перемістимо курсор у поле зовнішнього ключа й відпустимо кнопку. У результаті відкриється вікно Редагування зв'язків (рис. 2.11).
4. Увімкнемо прапорець Забезпечення цілісності даних. Після цього стануть доступними прапорці Каскадне оновлення пов'язаних полів і Каскадне видалення пов'язаних полів. Увімкнемо останній прапорець.

Увімкнення прапорця Забезпечення цілісності даних дає змогу зберегти цілісність даних. Якщо цей прапорець вимкнутий, то в таблиці можна додавати нові записи, змінювати ключові поля й вилучати пов'язані записи без попередження про порушення цілісності.

Сутність каскадного оновлення пов'язаних полів полягає в тому, що за будь-якої зміни первинного ключа в головній таблиці автоматично оновиться значення відповідного поля у всіх зв'язаних таблицях.

Сутність каскадного вилучення пов'язаних полів полягає в тому, що під час вилучення будь-якого запису з головної таблиці автоматично вилучаються зв'язані записи у зв'язаній таблиці. Отже, каскадне оновлення й каскадне вилучення прискорюють роботу з БД і сприяють підвищенню надійності її функціонування.

5. У вікні Редагування зв'язків натиснемо кнопку Створити, у результаті чого у вікні Звяз'ки з'явиться лінія зв'язку між певними полями таблиць (рис. 2.12). Збережемо БД.

У вікні Редагування зв'язків можна вилучити встановлений зв'язок за допомогою кнопки Скасувати. За допомогою кнопки Створити... можна відкрити нове вікно й установити зв'язок заново.

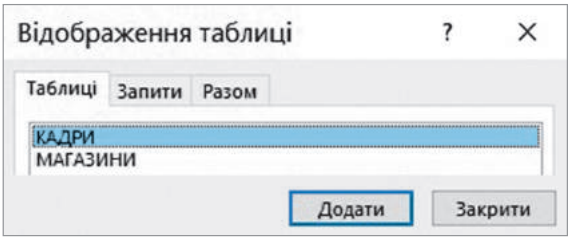


Рис. 2.10. Вікно з переліком таблиць БД atb

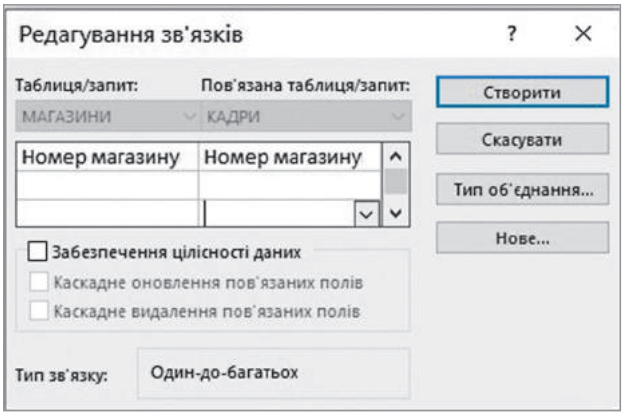


Рис. 2.11. Вікно **Редагування зв'язків**

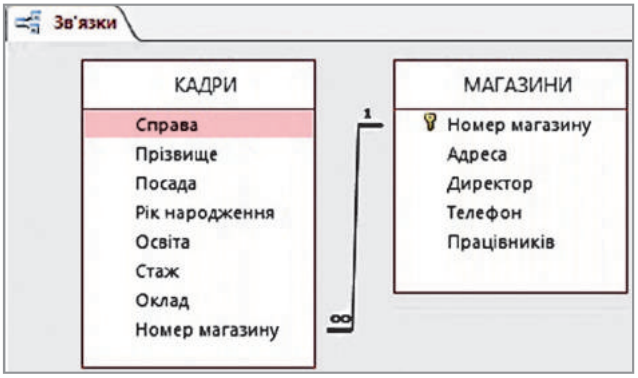


Рис. 2.12. Зв'язок типу **один-до-багатьох** між таблицями

6. Повернемося до вікна Редагування зв'язків (якщо в цей момент його немає на екрані, слід виконати команду Змінити зв'язки в області Знаряддя — і вікно відкриється). Далі натиснемо кнопку Тип об'єднання... З'явиться вікно Параметри об'єднання (рис. 2.13).
7. За замовчуванням установлюється перший тип об'єднання, який називають об'єднанням за еквівалентністю. Зазвичай розробники БД установлюють відношення за еквівалентністю. Натиснемо в цьому вікні кнопку ОК і закриємо вікно Редагування зв'язків.

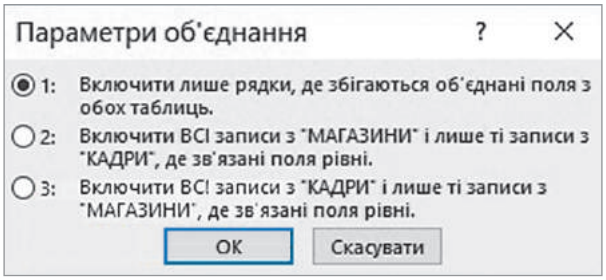


Рис. 2.13. Вікно для встановлення параметрів об'єднання таблиць

Щоб переглянути зв'язки між таблицями БД, необхідно на відкритій вкладці Знаряддя бази даних натиснути кнопку Зв'язки, активувати вкладку Конструктор і в розділі Зв'язок натиснути кнопку Усі зв'язки.

Якщо схема складна, можна приховати частину таблиць і зв'язків, вилучивши їх із вікна Зв'язки. Для цього потрібно виділити певну таблицю й натиснути кнопку Delete. При цьому зв'язки й таблиці вилучаються лише з вікна, фізично ж вони залишаються, тому в будь-який час їх можна відновити.

Щоб скасувати зв'язки між таблицями БД, необхідно встановити курсор на лінії зв'язку, натиснути кнопку миші, а потім — клавішу Delete. Можна також відкрити контекстне меню лінії зв'язку й виконати команду Видалити.

Запитання для перевірки знань

1. Що називають первинним ключем таблиці?
2. Які первинні ключі називають простими; складними?
3. Для чого використовується ключове поле типу **Лічильник**?
4. Який порядок створення ключового поля?
5. Що називають індексуванням таблиць?
6. Як можна скасувати зв'язок між таблицями?
7. З якою метою індексуються таблиці?
8. Для чого потрібно зв'язувати таблиці?
9. Опишіть порядок зв'язування таблиць.

Завдання для самостійного виконання

1. Створіть просту індексну таблицю для поля **Учнів** таблиці КЛАСИ й індексну таблицю для поля **Адреса** таблиці УЧНІ.
2. Створіть у таблиці КЛАСИ БД **skola** первинний ключ і збережіть таблицю. Переконайтеся, що ключ встановлено правильно.
3. Виконайте зв'язування таблиці КЛАСИ і таблиці УЧНІ. Доведіть, що зв'язок дійсно встановлений. Вилучіть зв'язок між таблицями й установіть його ще раз.

2.3. Введення, пошук і редагування даних у таблиці

Поясніть, у який спосіб можна здійснити пошук необхідних даних у таблиці.

Дані в таблиці можна вводити після створення їх структур. Існує два способи введення даних у таблиці: за допомогою форм і безпосереднє введення даних у таблиці.

Розглянемо алгоритм уведення даних на прикладі 1.

Приклад 1.

1. Відкриємо таблицю МАГАЗИНИ в режимі таблиці. Для цього двічі клацнемо кнопкою миші ім'я в області Усі об'єкти. Можна також скористатися контекстним меню цієї таблиці.
2. Уведемо дані першого запису таблиці МАГАЗИНИ, який позначено зірочкою. Щойно дані будуть уведені, курсор автоматично переміститься на наступний запис, що означає готовність до уведення даних у другий рядок. Слід пам'ятати, що вводити в поля можна лише ті типи даних, які збігаються з оголошеним типом поля.
3. У такому самому порядку введемо дані всіх інших записів таблиці й збережемо таблицю. Після введення останнього запису вміст таблиці набуде вигляду, як наведено на рис. 2.14.

Зверніть увагу на те, що записи виведені в іншому порядку, ніж уводилися (записи вводили в порядку, який подано у табл. 2.1).

На рис. 2.14 записи впорядковано в порядку значень ключового поля.

МАГАЗИНИ

Номер магазину	Адреса	Директор	Телефон	Працівникі	Клацніть, щоб додати
6	вул. Річкова, 24	Середа К. М.	234-67-92	15	
21	вул. Паркова, 33	Коцюба П. М.	234-54-63	20	
31	вул. Печерська, 21	Борзов А. С.	234-22-98	13	
0				0	

Запис: 4 з 4 | Без фільтра | Пошук

Рис. 2.14. Вміст таблиці МАГАЗИНИ

Під час уведення даних автоматично перевіряються такі типи даних: числові, грошові, дата і час, логічні. На вкладці Основне в групі Форматування тексту містяться елементи, за допомогою яких можна змінити розмір і накреслення символів та інші параметри.

Якщо на екрані не поміщаються всі записи, слід скористатися вертикальною смугою прокручування, а якщо не поміщаються всі поля — горизонтальною. Окремі поля можна розширити або звузити звичайним порядком.

У нижній частині вікна таблиці розміщено кнопки навігації для переміщення курсора в перший, сусідній або останній запис.

Для додавання нового запису в таблицю необхідно натиснути кнопку Створити запис на панелі навігації та ввести дані.

Навігацію в таблиці можна здійснювати за допомогою миші, смуг прокручування, сполучення деяких клавіш.

У такому самому порядку введемо дані в таблицю КАДРИ. Її вміст подано на рис. 2.15.

КАДРИ

Справа	Прізвище	Посада	Рік народження	Освіта	Стаж	Оклад	Номер магазину
105	Сокіл Т. Л.	касир	1960	середня	27	3500	6
109	Шрамко Т. Л.	диспетчер	1961	середня	24	4000	6
120	Рябко Р. П.	експерт	1981	вища	8	4200	21
111	Семко М. М.	диспетчер	1970	середня	16	4000	21
116	Раков Г. П.	аналітик	1965	вища	19	4500	21
132	Таран В. Д.	диспетчер	1973	вища	15	4000	31
115	Горошко Ф. Р.	диспетчер	1975	середня спеціальна	17	4000	31
0			0		0	0	0

Запис: 8 з 8 Без фільтра Пошук

Рис. 2.15. Вміст таблиці КАДРИ

Потрібний запис у таблиці можна знайти за значенням будь-якого її поля або за фрагментом його значення. Розглянемо пошук запису в таблиці на прикладі 2.

Приклад 2.

1. У таблиці КАДРИ в режимі таблиці встановимо курсор на поле, за значенням якого потрібно шукати запис (наприклад, на поле Прізвище), і натиснемо кнопку Знайти у розділі Пошук — відкриється вікно Пошук і заміна (рис. 2.16).
2. У вікно Пошук і заміна в поле Знайти введемо потрібне значення, наприклад Сокіл Т. Л., і натиснемо кнопку Знайти далі. Курсор встановиться на записі з прізвищем Сокіл Т. Л.

У полі Зіставити можна вибрати одне зі значень, що зменшує необхідність використання метасимволів:

- будь-яку частину поля (зразок може міститися всередині значення поля);
- усе поле (зразок без метасимволів має збігатися з усім значенням поля);
- початок поля (будуть знайдені тільки ті поля, які починаються зі зразка).

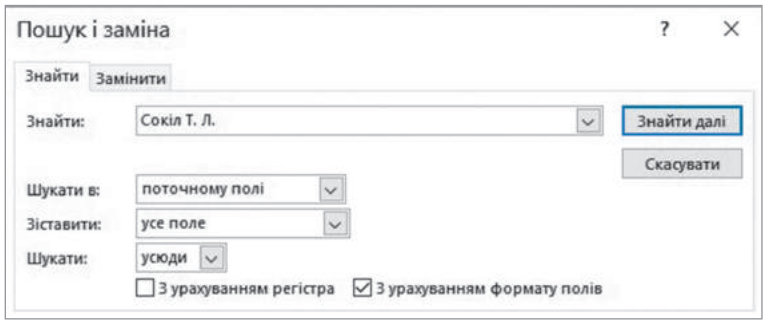


Рис. 2.16. Вікно для пошуку й заміни даних

У процесі введення даних у поле **Знайти** можна використовувати такі метасимволи:
* — довільна кількість будь-яких символів;
? — один довільний символ;
≠ — одна довільна фраза.

Якщо прапорець З урахуванням регістра встановлено, то зразок повинен точно відповідати шуканому фрагменту; якщо знято, то зразок можна вводити великими і малими буквами.

Якщо прапорець З урахуванням формату полів встановлено, то зразок повинен бути таким, як і шуканий фрагмент,

наприклад $345; якщо знято, то форматування не враховується, наприклад значення $345 буде знайдено і для зразка 345.

Після завершення пошуку на знайдений запис встановлюється курсор, і поле цього рядка висвітлюється іншим кольором. Оскільки в таблиці може бути кілька значень, що відшукуються, то для продовження пошуку необхідно ще раз натиснути кнопку Знайти далі. Для пошуку першого входження зразка можна використовувати поле Пошук. Пошук завершується після першого визначення рядка.

Знайдене значення можна змінювати, вводити нове. Поле типу Лічильник, заблоковані поля та поля, що обчислюються, змінювати не можна (приклад 3).

Записи з таблиці можна вирізати й копіювати до буфера обміну, за допомогою кнопки Вставити вставляти в іншу таблицю, а також у документи Word і Excel.

Розглянемо алгоритм вилучення запису з таблиці.

1. Виділити потрібний запис і натиснути кнопку Видалити на вкладці Основне. Відкриється меню цієї кнопки (рис. 2.17), у якому слід виконати команду Видалити запис.
2. У вікні, що відкриється (рис. 2.18), необхідно підтвердити або відмінити вилучення. Якщо потрібно вилучити весь запис, його слід виділити й натиснути кнопку Видалити, у меню (див. рис. 2.17) виконати команду Видалити запис.

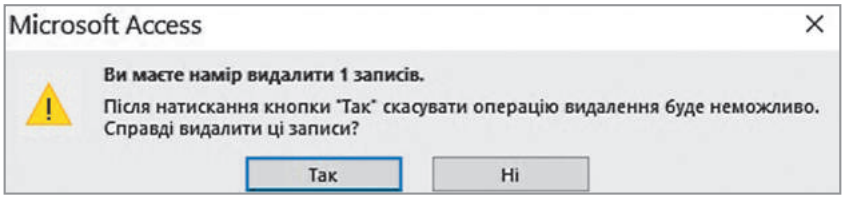


Рис. 2.18. Вікно підтвердження/відміни видалення запису

Зауважимо, що у зв'язаних таблицях зі встановленим прапорцем Забезпечення цілісності даних вилучити запис не вдається.

Приклад 3. Знайдемо в таблиці КАДРИ запис, у полі **Рік народження** якого є значення 1981. Буде виділено запис із прізвищем Рябко Р. П.

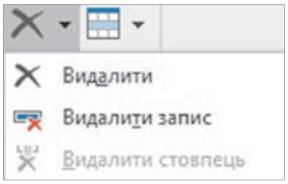


Рис. 2.17. Меню кнопки **Видалити**

Коли потрібно приховати деякі поля, їх слід виділити й у групі **Записи** виконати команду **Додатково → Приховати поля**. Щоб відновити приховані поля, слід виконати команду **Додатково → Відобразити поля** та у вікні, що відкриється, увімкнути прапорець відповідного поля.

Запитання для перевірки знань

1. Поясніть порядок уведення записів у таблицю.
2. За значенням якого поля впорядковуються записи за замовчуванням?
3. Як можна здійснити навігацію в таблиці?
4. Як здійснюється пошук запису в таблиці?
5. Як вилучити запис із таблиці?
6. Які дії слід виконати для додавання нового запису в таблицю?
7. З якою метою і як приховують поля?

Завдання для самостійного виконання

1. Уведіть у таблиці КЛАСИ й УЧНІ БД **skola** наведені в них дані.
2. Виконайте навігацію по таблиці УЧНІ за допомогою кнопок навігації.
3. Додайте в таблицю УЧНІ новий запис, а потім вилучте його. Збережіть таблицю.
4. Знайдіть у таблиці УЧНІ запис за прізвищем Пека П. О.
5. Використайте метасимволи для пошуку записів у таблиці УЧНІ за значенням поля **Адреса**.
6. Вилучте з таблиці УЧНІ будь-який запис, а потім відновіть його. Збережіть таблицю.

2.4. Сортування і фільтрування записів. Операції над таблицями

Сортування записів у таблиці можна реалізувати за значенням будь-якого поля. А що ви розумієте під терміном «фільтрація записів»?

Згадаємо, що за замовчуванням записи таблиці виводяться впорядкованими за значенням первинного ключа. Проте часто виникає необхідність отримати записи, упорядковані за значенням інших полів.

Сортування записів — це впорядкування записів за значеннями одного поля або кількох полів.

Для сортування **за значенням одного поля** треба його виділити й натиснути кнопку **За зростанням (А → Я)** або **За спаданням (Я → А)**. Можна також скористатися контекстним меню поля.

Для **сортування за значеннями кількох полів** необхідно ці поля виділити й скористатися одним із наведених далі способів.

Сортування записів виконується спочатку за значенням лівого виділеного поля. Якщо в ньому є поля, значення яких збігаються, то певні записи впорядковуються за значенням наступного поля. Наприклад, у результаті сортування записів За зростанням за значеннями полів Освіта і Стаж таблиці КАДРИ отримаємо розміщення записів, як наведено на рис. 2.19.

КАДРИ

Справа	Прізвище	Посада	Рік народження	Освіта	Стаж	Оклад	Номер магазину
120	Рябко Р. П.	експерт	1981	вища	8	4200	21
132	Таран В. Д.	диспетчер	1973	вища	15	4000	31
116	Раков Г. П.	аналітик	1965	вища	19	4500	21
111	Семко М. М.	диспетчер	1970	середня	16	4000	21
109	Шрамко Т. Л.	диспетчер	1961	середня	24	4000	6
105	Сокіл Т. Л.	касир	1960	середня	27	3500	6
115	Горошко Ф. Р.	диспетчер	1975	середня спеціальна	17	4000	31
0			0		0	0	0

Рис. 2.19. Таблиця КАДРИ, упорядкована за значенням полів **Освіта** і **Стаж**

Як бачимо з рис. 2.19, основне сортування виконано за зростанням значення поля Освіта. Записи з однаковим значенням упорядковано за зростанням значення поля Стаж (приклад 1).

Приклад 1. Якщо здійснити фільтрацію записів таблиці КАДРИ за значенням **більше або дорівнює 24** в полі **Стаж**, то отримаємо записи з прізвищами Сокіл Т. Л. і Шрамко Т. Л.

Фільтрування записів — це відбір із таблиці записів, які містять задане значення у вибраних полях.

Фільтрування можна виконати за виділенням і формою.

Фільтрування за виділенням — це відбір записів на основі значень поточного поля. Для його реалізації спочатку треба впорядкувати записи за значенням поля, яке використовується у фільтрації; встановити курсор на тому значенні поля, за яким буде виконуватися фільтрування; натиснути на кнопку Виділення в групі Сортування й фільтр та вибрати необхідну умову в меню, що відкриється (приклад 2).

Приклад 2. Відфільтруємо записи таблиці КАДРИ за значенням середня поля Освіта.

1. Упорядкуємо записи таблиці КАДРИ за значенням поля Освіта.
2. Встановимо курсор на назві середня цього поля і натиснемо кнопку Виділення в групі Сортування й фільтр. Відкриється меню з умовами (рис. 2.20).
3. Виберемо першу умову Дорівнює «середня». Відкриється таблиця, вміст якої наведено на рис. 2.21.

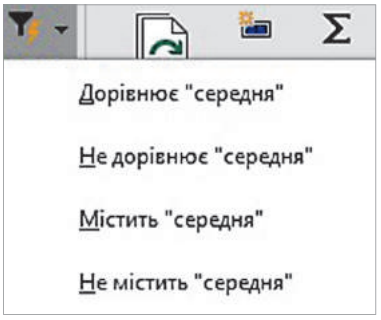


Рис. 2.20. Меню умов для фільтрації записів

У нижній частині таблиці висвітлиться кнопка з написом Відфільтровано (або Не відфільтровано). Це означає, що записи відфільтровано (або не відфільтровано). Натискаючи цю кнопку, можна вмикати й вимикати фільтрування записів. До відфільтрованих записів можна застосовувати ще кілька фільтрів, наприклад за значеннями полів Посада, Оклад та ін.

КАДРИ

Справа	Прізвище	Посада	Рік народження	Освіта	Стаж	Оклад	Номер магазину
109	Шрамко Т. Л.	диспетчер	1961	середня	24	4000	6
111	Семко М. М.	диспетчер	1970	середня	16	4000	21
105	Сокіл Т. Л.	касир	1960	середня	27	3500	6
*			0		0	0	0

Запис: 4 з 4 | Відфільтровано | Пошук

Рис. 2.21. Таблиця з відфільтрованими записами

Використовуючи фільтрування за формою, можна вводити критерії в поля таблиці умов. Розглянемо алгоритм відкриття таблиці умов на прикладі 3.

Приклад 3.

1. У розділі Сортування й фільтр відкриємо меню кнопки Параметри розширеного фільтра й виконаємо команду Розширений фільтр/сортування. Відкриється перелік полів таблиці, а в нижній частині вікна — таблиця конструктора.
2. У таблицю конструктора, наприклад, із відфільтрованої таблиці КАДРИ (див. рис. 2.21) перенесемо ім'я поля, у яке вводитиметься критерій. Для цього достатньо двічі клацнути його в таблиці. Перенесемо, наприклад, ім'я поля Стаж.
3. У таблиці конструктора в запис Критерії цього поля введемо умову, наприклад, >16. Далі відкриємо меню кнопки Параметри розширеного фільтра, у якому виконаємо команду Застосувати фільтр/сортування. Отримаємо записи (рис. 2.22).

КАДРИ | КАДРИФільтр1

Справа	Прізвище	Посада	Рік народження	Освіта	Стаж	Оклад	Номер магазину
109	Шрамко Т. Л.	диспетчер	1961	середня	24	4000	6
105	Сокіл Т. Л.	касир	1960	середня	27	3500	6
*			0		0	0	0

Запис: 3 з 3 | Відфільтровано | Пошук

Рис. 2.22. Відфільтровані записи відібрано за критерієм поля **Стаж >16**

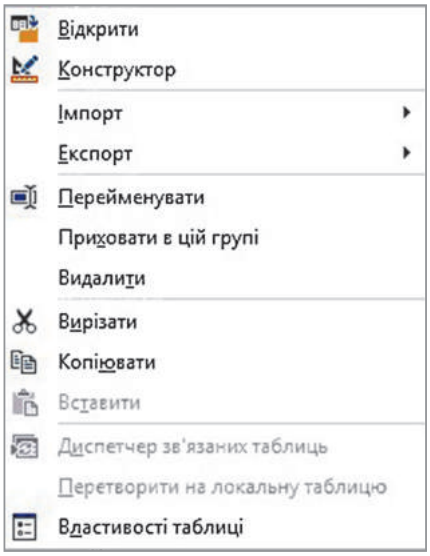


Рис. 2.23. Контекстне меню таблиці, відкрите в області **Усі об'єкти**

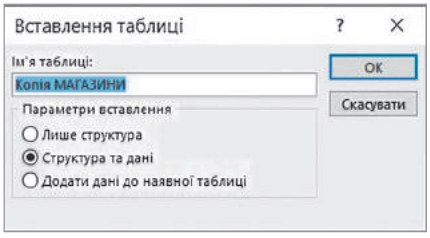


Рис. 2.24. Вікно для копіювання таблиці

У таблиці умов можна задати довільну кількість критеріїв фільтрування, які об'єднуються операторами І/АБО.

Над таблицями в середовищі Access можна виконувати операції перейменування, вилучення, копіювання та ін.

Перейменувати таблицю можна за допомогою її контекстного меню, яке відкривається шляхом клацання імені таблиці в області Усі об'єкти. Вміст меню наведено на рис. 2.23.

Для перейменування таблиці слід виконати такі дії.

Крок 1	Виконати команду **Перейменувати**
Крок 2	В області **Усі об'єкти** в текстове поле цієї таблиці ввести нове ім'я
Крок 3	Натиснути клавішу **Enter**. Слід пам'ятати, що після цього потрібно змінити це ім'я в усіх об'єктах БД (запитах, звітах та ін.)

Для вилучення таблиці можна також скористатися її контекстним меню й виконати в ньому команду Видалити.

Для копіювання таблиці в контекстному меню слід виконати команду Копіювати, ще раз викликати контекстне меню й виконати команду Вставити. Відкриється вікно Вставлення таблиці (рис. 2.24). Далі в поле Ім'я таблиці потрібно ввести нове ім'я та ввімкнути один із розташованих нижче перемикачів.

Якщо таблиця копіюється повністю, то слід увімкнути перемикач Структура та дані й натиснути кнопку ОК.

Запитання для перевірки знань

1. Які існують способи сортування записів?
2. Як упорядкувати записи за значенням одного поля?
3. Що називають фільтрацією записів?
4. Як упорядкувати записи за значеннями двох полів?
5. Які операції виконують над таблицями?
6. Як перейменувати таблицю?
7. Як здійснити фільтрування записів за виділенням?
8. Поясніть сутність фільтрації записів за формою.
9. Як скопіювати таблиці?
10. Як вилучити таблиці?

Завдання для самостійного виконання

1. Упорядкуйте записи таблиці УЧНІ БД **skola** за значенням поля **Зріст** (див. с. 18).
2. Упорядкуйте записи таблиці УЧНІ за значеннями полів **Прізвище** й **Історія**.
3. Виконайте фільтрування записів таблиці УЧНІ за одним зі значень поля **Улюблений предмет**.
4. Використайте фільтр **за формою** для фільтрування записів таблиці УЧНІ за одним зі значень поля **Дата народження**.
5. Надайте таблиці КЛАСИ нове ім'я. Відновіть ім'я КЛАСИ.
6. Виконайте копіювання таблиці УЧНІ й створіть на її основі нову таблицю з іменем ПЕРША.

3. Запити

3.1. Загальні відомості про запити

Пригадайте основне призначення запитів.

Запит — один із основних об'єктів БД Access. Головне його призначення полягає у відборі потрібних даних із таблиць, їх опрацюванні та поданні користувачеві у зручній формі.

Запит застосовується також для змінення даних у БД.

Створений запит можна зберігати з певним іменем і потім неодноразово виконувати. Якщо між першим і другим запусками запиту дані в таблицях змінилися, то в процесі другого його виконання будуть використовуватися оновлені дані.

Запити класифікують за багатьма ознаками. Розподіл запитів за основними ознаками наведено на рис. 3.1.

Запити не містять даних. Під час кожного нового виконання запиту формуються необхідні дані з тих таблиць, на основі яких його створено.

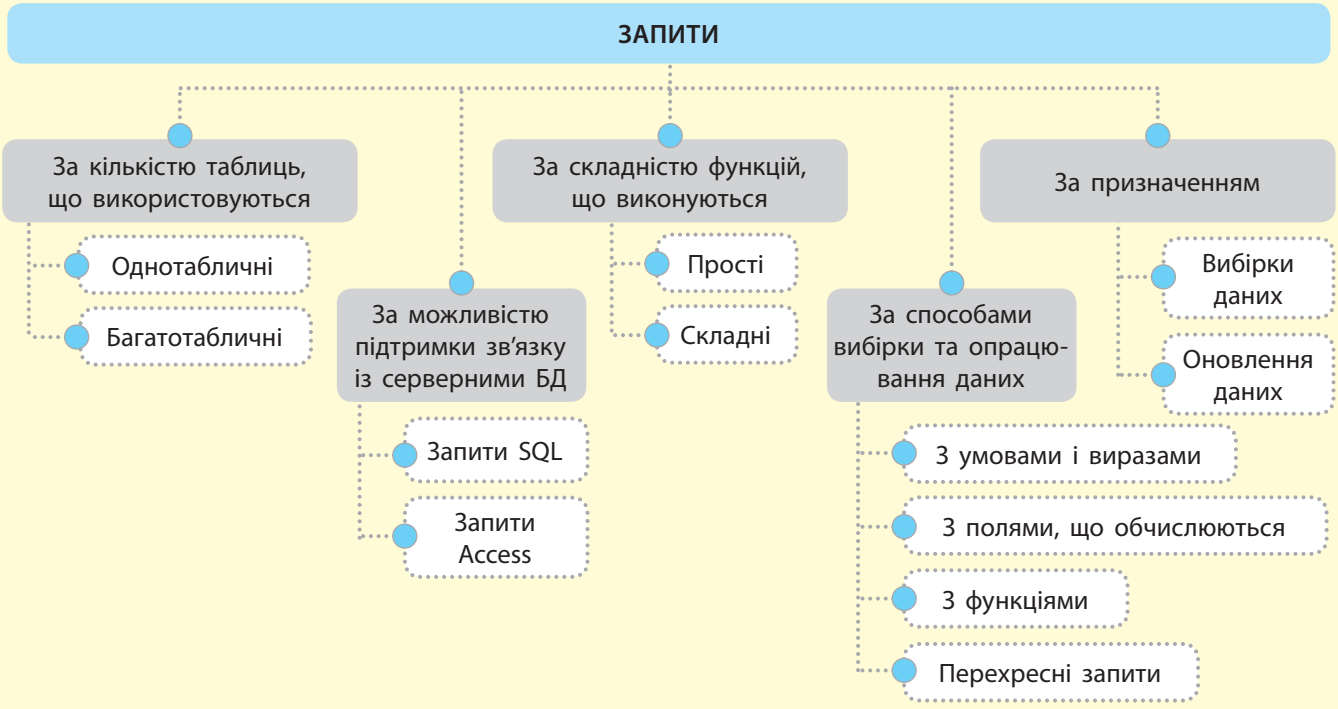


Рис. 3.1. Класифікація запитів

Запити, за допомогою яких вибирають дані з однієї таблиці, називають однотабличними.

Запити, за допомогою яких вибирають дані з кількох таблиць, називають багатотабличними.

Простими називають запити, за якими вибирають дані за критерієм одного поля однієї таблиці (приклад 1).

Складними називають запити, за якими вибирають дані за критеріями кількох полів із кількох таблиць (приклад 2).

Приклад 1. Приклади *простих* запитів: вибрати з таблиці МАГАЗИНИ ті номери магазинів, де кількість працівників більша за 13; вибрати з таблиці КАДРИ прізвища тих працівників, які народилися після 1979 року.

Приклад 2. Складний запит для таблиць МАГАЗИНИ і КАДРИ, за яким формуються дані, наведено в табл. 3.1. Тут із таблиць МАГАЗИНИ і КАДРИ відібрано прізвища осіб, які працюють диспетчерами в магазинах 21 і 31.

Таблиця 3.1. Результат виконання складного запиту

Магазин	Адреса	Прізвище	Посада
21	вул. Паркова, 33	Семко М. М.	диспетчер
31	вул. Печерська, 21	Таран В. Д.	диспетчер
31	вул. Печерська, 21	Горошко Ф. Р.	диспетчер

Американська корпорація Oracle (Oracle Corporation), заснована в 1977 році Ларрі Еллісоном, нині є найбільшим у світі розробником ПЗ для організацій. Oracle є абсолютним лідером на ринку розробки СУБД.

До однієї БД Access може бути розроблено кілька запитів. Кожен із них можна виконати в будь-який час, і кожен із них виконує чітко визначені функції. Запити можуть виконуватися самостійно, але найчастіше їх використовують як складові форм і звітів.

Як вам відомо, запити не містять даних, лише формують потрібні дані з таблиць. Наприклад, дані, наведені в табл. 3.1, не зберігаються, а формуються під час виконання запиту. Щоб зберегти дані, необхідно створити таблицю та скопіювати до неї ці дані. Описаний тип називають запитом на вибірку.

Запити на вибірку даних — запити, які забезпечують добір потрібних даних із таблиць. Такий тип запитів є одним із найбільш розповсюджених.

Разом із тим у Access використовуються й запити на змінення (оновлення даних) — запити, за допомогою яких здійснюється модифікування структури таблиць і змінення в них даних.

Найчастіше дані за допомогою запитів вибираються на основі критеріїв. Окрім того, система Access має набір убудованих функції, за допомогою яких дані можна вибрати з таблиць, а також опрацювати й узагальнити.

З опрацьованих даних можна створювати нові поля. Такий тип запитів називають запитами з полями, що обчислюються.

У запитах різних типів найчастіше реалізуються такі операції:

- вибір даних із вказаних полів на основі заданих критеріїв;
- упорядкування даних із таблиць за значеннями вказаних полів;
- побудова нової таблиці або діаграми з отриманих даних;
- опрацювання вибраних із таблиць даних за допомогою вбудованих функцій;
- використання отриманих за допомогою запиту даних як джерела для інших запитів;
- додавання даних, отриманих за допомогою запитів, до інших таблиць;
- обмін даними з іншими БД, а також текстовим редактором Word і таблицями Excel.

У запитах можуть використовуватися специфічні оператори.

- Рядкові оператори:

  Like (які збігаються/відповідність);
  Not Like (які не збігаються/невідповідність);
  об'єднання рядків (&).

Вирази в критеріях, що застосовуються в запитах, будуються на основі звичайних арифметичних операцій, операцій порівняння й логічних операцій (And, Or, Xor, Not).

Оператори Like і Not Like використовуються для порівняння двох рядкових виразів. При цьому перевіряється, чи збігаються ці вирази, і залежно від результату повертаються значення Так, Ні або Null.

Оператор Like має таку структуру: <ім'я поля> Like <зразок> (приклад 3).

- Оператори списку й діапазону:
  In (входження в список);
  Is (наявність значення);
  Between...And (входження в діапазон).

За допомогою оператора In перевіряється, чи збігається значення поля з одним зі значень списку. Якщо збігається, повертається значення Так, інакше — Ні (приклад 4).

Оператор Is використовується тільки з ключовим словом Null для з'ясування, чи містить об'єкт будь-яке значення. Повертається значення Так, якщо вираз порожній (не містить жодного значення).

Оператор Between...And має таку структуру: <ім'я поля> Between <нижня межа> And <верхня межа>. Повертається значення Так, якщо значення поля знаходяться між значеннями <нижня межа> і <верхня межа> (приклад 5).

Критерії поділяються на прості й складні.

Прості критерії зазвичай містяться лише в одному полі, складні критерії — у кількох полях. Наприклад, вибрати з таблиць МАГАЗИНИ і КАДРИ прізвища працівників магазинів, які народилися у період з 1961 до 1975 років і працюють диспетчерами або аналітиками.

Далі буде стисло описано методику створення усіх типів запитів (див. рис. 3.1), крім запитів SQL, для розроблення яких потрібно володіти мовою SQL.

Приклад 3. Для таблиці КАДРИ вираз **<Прізвище Like "Раков Г. П."**> повертає значення **Так**, оскільки поле **Прізвище** містить значення **Раков Г. П.**

Приклад 4. Для таблиці КАДРИ оператор **<Прізвище In ('Семко Н. Н.', 'Горошко Ф. Р'**)> повертає значення **Так**, тому що в цьому полі є зазначені прізвища.

Приклад 5. Для таблиці КАДРИ за допомогою оператора **[Рік народження] Between 1965 And 1973** повертається значення **Так**, оскільки в цьому полі є зазначені діапазони.

Запитання для перевірки знань

1. Назвіть основне призначення запитів.
2. Які запити називають простими?
3. Опишіть, як класифікують запити за призначенням.
4. Назвіть основні класифікаційні ознаки запитів.
5. Наведіть приклад простого запиту.
6. Які функції виконують запити на вибірку?
7. Назвіть операції, які найчастіше реалізуються в запитах.
8. Які існують оператори списку й діапазону?
9. Наведіть приклад використання операторів **Like** і **Not Like**.

Завдання для самостійного виконання

1. Накресліть класифікацію запитів за призначенням.
2. Наведіть приклад простого запиту для таблиці КАДРИ.
3. Наведіть приклад простого запиту для таблиці МАГАЗИНИ.
4. Наведіть приклад використання оператора **In** для таблиці МАГАЗИНИ.
5. Наведіть приклад складного запиту для таблиць МАГАЗИНИ і КАДРИ.
6. Накресліть класифікацію запитів за основними ознаками.

3.2. Запити на вибірку даних

Поміркуйте, що необхідно зробити для одночасного отримання потрібних даних із кількох таблиць БД.

Запити на вибірку даних — це запити, які забезпечують вибір необхідних даних із однієї або кількох таблиць.

Розглянемо загальний порядок створення простого запиту на вибірку (запиту для однієї таблиці) та приклад 1.

Крок 1	Відкрити БД, активувати вкладку **Створення** й у розділі **Запити** клацнути кнопку **Макет запиту**, який фактично є конструктором запиту. У результаті відкриються вікно конструктора запиту (вікно **Запит1**) і вікно **Відображення таблиці,** у якому містяться імена всіх таблиць цієї БД. На панелі інструментів вкладки **Конструктор** з'явилася група кнопок **Тип запиту**, у якій виділено кнопку **Вибір**. Це означає, що запит на вибірку створюється за замовчуванням. Якщо створюватимуться інші типи запитів, то потрібно вмикати відповідну кнопку в цій групі.
Крок 2	Вибрати у вікні **Відображення таблиці** необхідну таблицю — відкриється перелік її полів.
Крок 3	Створити запит на основі вмісту таблиці.

Приклад 1. Створити простий запит з іменем Запит_1, за допомогою якого з таблиці КАДРИ виводяться дані про співробітників зі стажем понад 16 років. Результуючий набір записів повинен містити такі поля: Справа, Прізвище, Рік народження, Стаж, Номер магазину.

1. Для створення запиту відкриємо БД atb і виконаємо команду Створення → Макет запиту. Відкриються вікно конструктора запитів і вікно Відображення таблиці, у якому виберемо таблицю КАДРИ. Для цього встановимо курсор на імені цієї таблиці й клацнемо кнопку Додати. Після цього вікно Відображення таблиці можна буде закрити.
2. У записі Поле таблиці конструктора запитів послідовно розмістимо зазначені імена полів (Справа, Прізвище, Рік народження, Стаж, Номер магазину) таблиці КАДРИ (рис. 3.2). Для цього достатньо двічі клацнути кнопкою миші на імені певного поля цієї таблиці.

Запис Сортування використовується для сортування даних у таблиці, яку буде отримано після виконання запиту. Сортувати дані можна за значенням кількох полів. Прапорець, установлений на перетині запису Відображення й певного поля, означає, що це поле буде виведено на екран, інакше воно виводитися не буде. Запис Критерії призначено для запису виразу, на основі якого відбираються записи. Запис Або слугує для визначення додаткової умови відбору записів.

3. У запис Критерії поля Стаж уведемо вираз >16. Збережемо запит, для чого на панелі швидкого доступу натиснемо кнопку Зберегти й у вікні, що відкриється, уведемо ім'я Запит_1 і натиснемо кнопку ОК. У результаті ім'я цього запиту з'явиться в області переходів.
4. Виконаємо запит. Для цього на стрічці в групі Результати натиснемо кнопку Запуск!. Отримаємо результат, як подано на рис. 3.3.

5. Для закриття запиту відкриємо його контекстне меню і виконаємо команду Закрити.

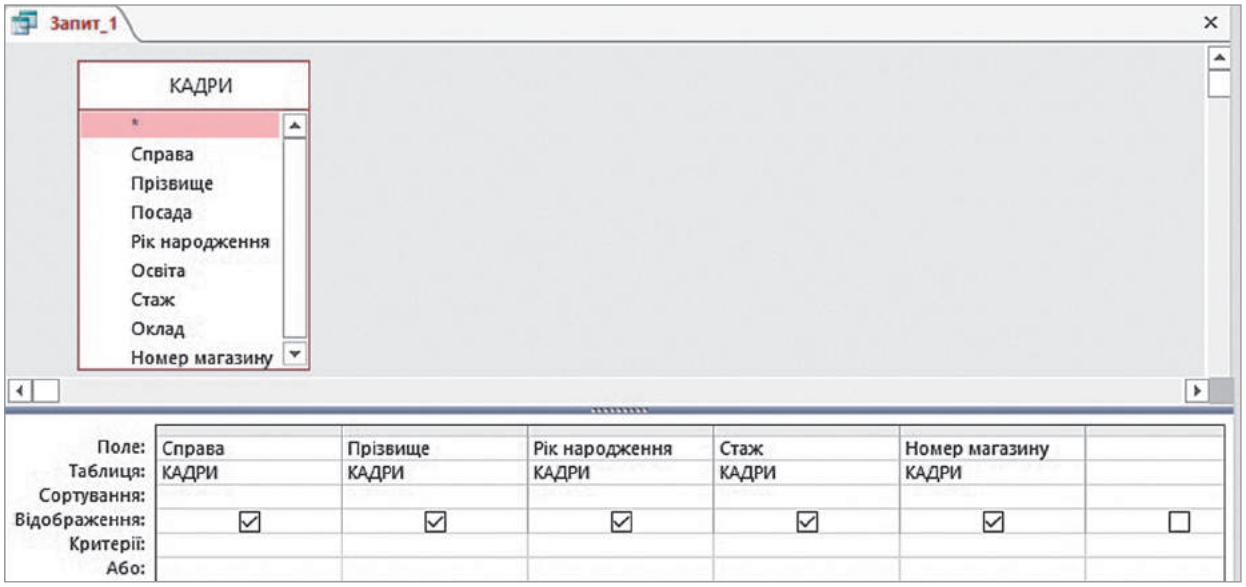


Рис. 3.2. Вікно конструктора запитів

Запит_1

Справа	Прізвище	Рік народження	Стаж	Номер магазину
105	Сокіл Т. Л.	1960	27	6
115	Горошко Ф. Р.	1975	17	31
116	Раков Г. П.	1965	19	21
109	Шрамко Т. Л.	1961	24	6
		0	0	0

Рис. 3.3. Дані про працівників магазинів зі стажем роботи понад 16 років

Створений запит можна перейменовувати й редагувати.

Щоб перейменувати запит, слід відкрити його контекстне меню й виконати команду Перейменувати. Ім'я цього запиту в області переходів буде виділено прямокутником іншого кольору. У поле слід увести нове ім'я й натиснути клавішу Enter.

У процесі редагування запиту можна виконувати такі дії:

- додавати поля в запит із таблиці;
- вилучати поля;
- додавати нові поля;
- змінювати розміри полів;
- змінювати порядок розміщення полів;
- змінювати критерії відбору записів, порядок їх сортування й порядок виведення (невиведення) полів;
- перейменовувати поля запиту;
- вилучати таблиці із запиту (для багатотабличних запитів).

Для виконання цих операцій запит потрібно відкрити в режимі конструктора. Їх виконують так само, як і аналогічні операції в таблицях (приклад 2).

Методика **створення запиту для кількох таблиць** багато в чому схожа на методику створення запиту для однієї таблиці. Лише слід ураховувати, що таблиці обов'язково повинні мати між собою зв'язок.

Приклад 2. Створити Запит_2, за допомогою якого в результуючу таблицю виводяться прізвища диспетчерів із полями Номер магазину і Телефон із таблиці МАГАЗИНИ, а з таблиці КАДРИ — поля Прізвище, Посада й Освіта. Упорядкувати записи за прізвищами працівників в алфавітному порядку.

1. У відкритій БД atb активуємо вкладку Створення й виконаємо команду Макет запиту. Додамо у вікно конструктора запиту обидві таблиці й закриємо вікно Відображення таблиць.
2. У запис Поле таблиці конструктора запиту перенесемо поля Номер магазину й Телефон таблиці МАГАЗИНИ і поля Прізвище, Посада й Освіта таблиці КАДРИ.
3. У запис Критерії поля Посада введемо назву професії диспетчер, а в записі Сортування поля Прізвище встановимо значення За зростанням.
4. Збережемо запит з іменем Запит_2 і виконаємо його. На екрані повинні з'явитися записи, як наведено на рис. 3.4.

Запит_2

Номер магазину	Телефон	Прізвище	Посада	Освіта
31	234-22-98	Горошко Ф. Р.	диспетчер	середня спеціальна
21	234=54-63	Семко М. М.	диспетчер	середня
31	234-22-98	Таран В. Д.	диспетчер	вища
6	234-67-92	Шрамко Т. Л.	диспетчер	середня

Рис. 3.4. Дані про диспетчерів, вибрані з двох таблиць

Запитання для перевірки знань

1. Які запити називають запитами на вибірку даних?
2. Як зберігається запит?
3. Опишіть режими, у яких можна відкрити запит.
4. Для чого призначено запис **Критерії** таблиці конструктора запиту?
5. Як можна змінити порядок розміщення полів у запиті?
6. Поясніть різницю між створенням запиту для однієї таблиці та кількох таблиць.
7. Як створити запит на вибірку даних?
8. Які дії можна виконувати під час редагування запиту?

Завдання для самостійного виконання

1. Створіть **Запит41** на основі таблиці УЧНІ, за допомогою якого вибираються прізвища учнів, улюбленим предметом яких є історія. Записи повинні мати поля **Прізвище**, **Клас**, **Улюблений предмет**.
2. Створіть **Запит42** на основі таблиці УЧНІ, за допомогою якого вибираються прізвища учнів, зріст яких більший від 170.
3. Створіть **Запит43** на основі таблиці УЧНІ, за допомогою якого вибираються прізвища учнів, улюбленими предметами яких є математика та інформатика.
4. Створіть **Запит44** на основі таблиць КЛАСИ й УЧНІ, за допомогою якого вибираються прізвища класних керівників учнів, які з інформатики та історії мають бали, більші від 10.
5. Створіть **Запит45** на основі таблиці УЧНІ, за допомогою якого вибираються прізвища учнів, які народилися у 2002 році.
6. Створіть **Запит46** на основі таблиць КЛАСИ й УЧНІ, за допомогою якого для класів, у яких учнів більше ніж 24, вибираються прізвища учнів, які з історії мають більше від 9 балів.

3.3. Запити з функціями і з полями, що обчислюються

Назвіть переваги використання стандартних функцій у запитах.

Ми вже розглянули запити, за допомогою яких із таблиць можна вибрати необхідні дані за певними критеріями. Отримані дані можна також опрацьовувати (наприклад, обчислити середнє значення поля, знайти серед знайдених запис із мінімальним значенням певного поля тощо). Так, для таблиці КАДРИ можна обчислити середній стаж роботи працівників із вищою освітою.

У системі Access є вбудовані функції, що дають змогу узагальнити дані деяких полів і полегшити опрацювання даних.

Запити, у яких використовуються такі функції, називають по-різному, наприклад підсумковими запитами. Але найчастіше їх називають запитами з функціями.

У системі Access 2.16 існує два способи використання перелічених функцій:

- до запиту, відкритого в режимі таблиці, додається запис підсумків, у якому для кожного поля може використовуватись одна з функцій;
- у режимі конструктора створюється підсумковий запит, у якому обчислюються проміжні підсумки за групами записів.

Розглянемо на прикладі 1 перший спосіб.

Деякі функції системи Access:

- **Sum** (Сума) — обчислює суму значень елементів поля;
- **Avg** (Середнє) — обчислює середнє значення поля;
- **Max/Min** (Максимум/Мінімум) — повертає елемент із максимальним/мінімальним значенням поля;
- **Count** (Кількість) — підраховує кількість записів за значенням поля.

Приклад 1. Створити Запит_4, за допомогою якого з таблиці КАДРИ вибираються записи про співробітників, які народилися після 1961 року і мають стаж понад 15 років. Результуючі записи повинні містити поля: Прізвище, Посада, Рік народження, Стаж і Оклад. Підрахувати кількість результуючих записів за значенням поля Прізвище й обчислити загальну суму окладів цих осіб.

1. Створимо в режимі конструктора звичайний запит на вибірку (рис. 3.5).
2. Збережемо запит з іменем Запит_4 і виконаємо його. Результат виконання запиту наведено на рис. 3.6.
3. На вкладці Основне в групі Записи натиснемо кнопку Підсумки (∑). Під останнім записом таблиці (рис. 3.6) з'явиться новий запис Підсумок. У цьому записі клацнемо поле Прізвище та в списку, що відкриється, виберемо функцію Кількість.
4. Аналогічно в цьому самому записі поля Оклад виберемо функцію Сума.

   У результаті отримаємо результат, як наведено на рис. 3.7.
5. Для збереження внесених змін ще раз клацнемо кнопку Зберегти.

Поле:	Прізвище	Посада	Рік народження	Стаж	Оклад
Таблиця:	КАДРИ	КАДРИ	КАДРИ	КАДРИ	КАДРИ
Підсумок:	Групування за	Групування за	Групування за	Групування за	Групування за
Сортування:					
Відображення:	☑	☑	☑	☑	☑
Критерії:			>1961	>15	
Або:					

Рис. 3.5. Приклад запиту на вибірку даних у режимі конструктора

Запит_4

Прізвище	Посада	Рік народження	Стаж	Оклад
Семко М. М.	диспетчер	1970	16	4000
Горошко Ф. Р.	диспетчер	1975	17	4000
Раков Г. П.	аналітик	1965	19	4500
*		0	0	0

Рис. 3.6. Результат виконання запиту на вибірку даних

Запит_4

Прізвище	Посада	Рік народження	Стаж	Оклад
Семко М. М.	диспетчер	1970	16	4000
Горошко Ф. Р.	диспетчер	1975	17	4000
Раков Г. П.	аналітик	1965	19	4500
*		0	0	0
3				12500

Рис. 3.7. Запит у режимі таблиці з підсумковим записом

Розглянемо на прикладі 2 другий спосіб.

Приклад 2.

1. Створимо звичайний запит на вибірку в режимі конструктора, наприклад запит, за допомогою якого з таблиці КАДРИ вибираються прізвища працівників з окладом понад 4000 грн і підраховується їх кількість. Результуючий перелік записів має містити поля Справа, Прізвище, Стаж, Оклад.
2. Уведемо в рядок Критерії поля Оклад вираз >4000.
3. На вкладці Конструктор відкриємо меню кнопки Відображення або приховання й виконаємо команду Підсумки (∑). У конструкторі запиту з'явиться рядок Підсумок, а в кожному полі цього запису буде зазначено Групування за.
4. У записі Підсумок клацнемо те поле, за яким потрібно виконати підрахунок кількості записів (наприклад, поле Справа).

   У списку, що відкриється, виберемо функцію Кількість (рис. 3.8).
5. Збережемо запит та виконаємо.

Поле:	Справа	Прізвище	Стаж	Оклад		
Таблиця:	КАДРИ	КАДРИ	КАДРИ	КАДРИ		
Підсумок:	Кількість	Групування за	Групування за	Групування за		
Сортування:						
Відображення:		☑	☑	☑	☐	☐
Критерії:				>4000		
Або:						

Групування за
Сума
Середнє
Min
Максимум
Кількість
StDev
Var
Перший
Останнє
Вираз
Розташування

Рис. 3.8. Приклад підсумкового запиту в режимі конструктора

Система Access 2016 дозволяє створювати запити з полями, що обчислюються. Таких полів у запиті може бути кілька.

Запити з полями, що обчислюються, — це запити, які дозволяють виводити в результуючий набір записів не лише поля таблиць, а й нові поля, які створює сам користувач.

У запитах із полями, що обчислюються, містяться дані, отримані під час обчислення даних полів таблиць. Наприклад, на основі даних таблиці КАДРИ в результуючий набір записів можна ввести поле Доплата, у якому обчислюється доплата до окладу залежно від стажу працівника (приклад 3).

Френсіс Елізабет Аллен — американська вчена, піонерка в галузі оптимізації компіляторів, перша жінка, яка отримала премію Тюрінга.

Приклад 3.

Припустимо, що за кожен рік стажу понад 5 років працівники отримують надбавку у розмірі 1 % від посадового окладу. Тоді надбавку можна обчислити за формулою:

Доплата = Оклад*(Стаж–5)/100.

На основі таблиці КАДРИ створимо запит з іменем Запит_5, за допомогою якого виводяться всі записи таблиці з полями Прізвище, Стаж, Оклад і Доплата, значення якого обчислюється за наведеною формулою. Порядок створення запиту такого типу несуттєво відрізняється від порядку звичайних запитів (приклад 4).

Приклад 4.

1. Відкриємо БД atb, активуємо вкладку Створення й клацнемо кнопку Макет запиту. Із таблиці КАДРИ перенесемо в конструктор запиту поля Прізвище, Стаж, Оклад, а в наступне поле введемо вираз: Доплата:[Оклад]*([Стаж]–5)/100. Зверніть увагу на те, що імена полів, які входять у вираз, беруться у квадратні дужки.
2. Установимо в записі Сортування поля Прізвище значення За зростанням для того, щоб прізвища виводилися в алфавітному порядку. Створений запит зображено на рис. 3.9.
3. Збережемо запит з іменем Запит_5 (згадаємо, що для цього потрібно натиснути кнопку Зберегти, у вікні, що відкриється, ввести ім'я запиту й клацнути кнопку ОК). У результаті виконання запиту має з'явитися результат, як наведено на рис. 3.10.
4. Закриємо Запит_5.

Поле:	Прізвище	Стаж	Оклад	Доплата: [Оклад]*([Стаж]-5)/100
Таблиця:	КАДРИ	КАДРИ	КАДРИ	
Сортування:	За зростанням			
Відображення:	☑	☑	☑	☑
Критерії:				
Або:				

Рис. 3.9. Запит із полем, що обчислюється

Запит_5

Прізвище	Стаж	Оклад	Доплата
Горошко Ф. Р.	17	4000	480
Раков Г. П.	19	4500	630
Рябко Р. П.	8	4200	126
Семко М. М.	16	4000	440
Сокіл Т. Л.	27	3500	770
Таран В. Д.	15	4000	400
Шрамко Т. Л.	24	4000	760
	0	0	

Рис. 3.10. Результат виконання запиту з полем, що обчислюється

Запитання для перевірки знань

1. Назвіть основні вбудовані функції для опрацювання даних у запиті.
2. Які запити називають запитами з полями, що обчислюються?
3. Яке призначення має функція **Count**(Кількість)?
4. Назвіть способи використання вбудованих функцій у запитах.
5. Опишіть порядок створення запитів із функціями.
6. Як додати до запиту підсумковий запис у режимі таблиці?
7. Поясніть порядок створення підсумкового запиту.
8. Поясніть порядок створення запиту з полями, що обчислюються.

Завдання для самостійного виконання

1. Створіть **Запит51** на основі таблиці УЧНІ, за допомогою якого підраховується кількість учнів, улюбленими предметами яких є географія та фізика.
   Записи повинні містити поля **Прізвище**, **Клас**, **Улюблений предмет**.
2. Створіть **Запит52** на основі таблиці УЧНІ, за допомогою якого обчислюється середній бал успішності учнів 11 класів окремо з інформатики й окремо з історії.
3. Створіть **Запит53** на основі таблиці УЧНІ, за допомогою якого обчислюється кількість учнів, які мешкають на вул. Лугова.
4. Створіть **Запит54** на основі таблиць КЛАСИ й УЧНІ, за допомогою якого обчислюється спільний середній бал успішності з інформатики та історії учнів класу, яким керує Дерев'янко Н. С.
5. Створіть **Запит55** на основі таблиці УЧНІ, за допомогою якого обчислюється середній зріст учнів кожного класу.
6. Створіть **Запит56** на основі таблиць КЛАСИ й УЧНІ, за допомогою якого для учнів 10 класу, улюбленим предметом яких є інформатика, обчислюється різниця середнього бала успішності з інформатики та історії.

3.4. Запити з параметрами. Перехресні запити

Ви вже знайомі з деякими типами запитів. Які ще функції доцільно використовувати для виконання запитів?

Ми вже розглядали запити з постійними критеріями, тобто запити, під час повторного виконання яких критерій відбору записів не змінювався. На практиці ж часто виникає потреба у зміненні цих критеріїв. Наприклад, під час першого виконання запиту слід вибрати із таблиці КАДРИ прізвища диспетчерів зі стажем роботи понад 10 років, а під час другого — прізвища диспетчерів зі стажем роботи понад 15 років. Такі дії можна виконувати за допомогою запитів із параметрами.

Запит із параметрами — це запит, у процесі виконання якого пропонується ввести деякі дані, наприклад умову, яку потрібно вставити в поле. Його ще називають *запитом зі змінними критеріями*.

За запитами з параметрами на початку їх виконання на екран виводяться повідомлення про необхідність уведення нового критерію (виразу). Методика створення запиту такого типу несуттєво відрізняється від методики створення звичайного запиту.

Розглянемо порядок створення такого запиту на прикладі 1.

Пошукова система Google була створена у 1998 році. За перші два роки її існування у БД цієї пошукової системи було зібрано дані про мільярд веб-сторінок, а у 2008 році — близько трильйона.

Приклад 1.

Розробити запит з іменем Запит_6, за допомогою якого з БД atb вибиратимуться прізвища працівників за посадами диспетчер і експерт із магазинів, номери яких уводяться при виконанні запиту. Результуючі записи мають містити поля Номер магазину й Адреса таблиці МАГАЗИНИ і поля Прізвище й Посада таблиці КАДРИ.

Порядок створення запиту може бути таким.

1. Відкриємо БД atb і виконаємо команду Створення → Макет запиту. Виділимо обидві таблиці, клацнемо кнопку Додати й закриємо вікно Відображення таблиці.
2. Із таблиці МАГАЗИНИ перенесемо в таблицю конструктора поля Номер магазину й Адреса, а з таблиці КАДРИ — поля Прізвище й Посада.
3. В умові завдання визначено, що за допомогою запиту мають відбиратися записи тільки за посадами диспетчер і експерт, тобто ця умова є незмінною. Тому в клітинку на перетині запису Критерії і поля Посада уводимо вираз "диспетчер" Or "експерт".

Щоразу після запуску при виконанні запиту користувач може вводити будь-який номер магазину. Тому на перетині запису Критерії та поля Номер магазину можна увести, наприклад, текст [У якому магазині?]. Головне, щоб текст містився у квадратних дужках.

У результаті отримаємо запит у режимі конструктора, як наведено на рис. 3.11.

Поле:	Номер магазину	Адреса	Прізвище	Посада
Таблиця:	МАГАЗИНИ	МАГАЗИНИ	КАДРИ	КАДРИ
Сортування:				
Відображення:	☑	☑	☑	☑
Критерії:	[У якому магазині?]			"диспетчер" Or "експерт"
Або:				

Рис. 3.11. Запит із параметром

4. Збережемо й виконаємо Запит_6. На екрані висвітлиться вікно Введення значення параметра із запитанням У якому магазині? (рис. 3.12).

Уведемо, наприклад, номер магазину 21, клацнемо кнопку ОК. Отримаємо результат, як наведено на рис. 3.13.

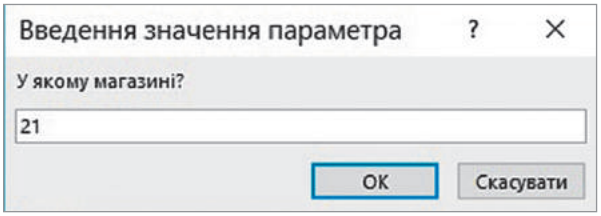


Рис. 3.12. Вікно введення значення параметра

Запит_6

Номер магазину	Адреса	Прізвище	Посада
21	вул. Паркова, 33	Рябко Р. П.	експерт
21	вул. Паркова, 33	Семко М. М.	диспетчер

Рис. 3.13. Результат виконання запиту з параметром

Проаналізуємо вміст таблиці КАДРИ й переконаємося, що дійсно в магазині з номером 21 на посадах диспетчера й експерта працюють відповідно Семко М. М. і Рябко Р. П.

Якщо потрібно здійснити пошук прізвищ диспетчерів і експертів у кількох магазинах, то в запис Критерії поля Номер магазину необхідно ввести іншу умову, а саме — діапазон номерів магазинів, наприклад:

>[Більше якого?] And <[Менше якого?].

У процесі виконання запиту спочатку з'явиться повідомлення Більше якого?, а потім — Менше якого?, на які потрібно дати певну відповідь.

Один запит може містити декілька параметрів у різних полях.

Для **створення перехресного запиту** потрібно використати щонайменше три поля:
- поле для визначення заголовка записів;
- поле для визначення заголовка полів;
- поле для вибору значень, над якими будуть виконуватися обчислення.

Перехресний запит — це запит на вибірку даних із можливостями групування записів.

Групування можна виконувати як за значеннями полів, так і за значеннями записів. Наприклад, із таблиці КАДРИ можна отримати кількість працівників на всіх посадах у кожному магазині. Як заголовки полів можуть бути використані й деякі вирази.

У режимі конструктора перехресний запит спочатку створюється як звичайний запит на вибірку даних, а потім установлюється режим перехресного запиту.

Розглянемо порядок створення перехресного запиту на прикладі 2.

Приклад 2. Створити перехресний запит, за допомогою якого підраховуються працівники на кожній посаді в кожному магазині.

1. Створимо звичайний запит на вибірку даних у режимі конструктора. Для цього виконаємо команди Створити → Макет запиту → Додати → Закрити. Перенесемо в таблицю конструктора запиту поля Посада й Номер магазину. Виконаємо запит і переконаємося, що він функціонує правильно.
2. Перемкнемо запит у режим конструктора і перетворимо в тип перехресний. У групі Тип запиту клацнемо кнопку Перехресний. У таблиці конструктора запиту з'являться записи Підсумок і Перехресний.

У записі Підсумок обох полів не змінюємо значення Групування за. Клацнемо клітинку на перетині поля Посада й запису Перехресний і виберемо зі списку, що розкриється, Заголовок рядка, а в полі Номер магазину цього самого запису — Заголовок стовпця.

3. У третє поле таблиці конструктора запиту перенесемо поле Номер магазину, у записі Підсумок якого встановимо функцію Кількість, а в записі Перехресний — Значення.

У записі Сортування першого поля можна встановити потрібне сортування записів. Таблиця конструктора запиту матиме вміст, як наведено на рис. 3.14.

Поле:	Посада	Номер магазину	Номер магазину
Таблиця:	КАДРИ	КАДРИ	КАДРИ
Підсумок:	Групування за	Групування за	Кількість
Перехресний:	Заголовок рядка	Заголовок стовпця	Значення
Сортування:			Заголовок рядка
Критерії:			Заголовок стовпця
Або:			Значення
			(не відображається)

Рис. 3.14. Таблиця конструктора перехресного запиту

Запитання для перевірки знань

1. Які запити називають запитами з параметрами?
2. Які поля обов'язково використовують у перехресних запитах?
3. Наведіть приклад запиту з параметрами.
4. Поясніть сутність перехресного запиту.
5. Поясніть порядок створення запиту з параметрами.
6. Опишіть порядок створення перехресного запиту.

Завдання для самостійного виконання

1. Створіть **Запит61** із параметрами на основі таблиці УЧНІ, за допомогою якого виводяться прізвища учнів із різними улюбленими шкільними предметами, назви яких уводяться в процесі виконання запиту.
2. Створіть **Запит62** із параметрами на основі таблиці УЧНІ, за допомогою якого виводяться прізвища учнів різного зросту, значення яких уводяться в процесі виконання запиту.
3. Створіть перехресний **Запит63**, за допомогою якого підраховується кількість учнів, які мають певний улюблений предмет, у всіх класах.
4. Створіть перехресний **Запит64**, за допомогою якого обчислюється кількість учнів окремо в кожному класі, зріст яких більший за 163 см.
5. Створіть **Запит65** із параметрами на основі таблиці УЧНІ, за допомогою якого виводяться прізвища учнів, їхній улюблений предмет і номер класу, що вводяться під час виконання запиту.
6. Створіть перехресний **Запит66**, за допомогою якого обчислюється кількість учнів, які мають певний улюблений предмет, у кожному класі.

3.5. Запити на змінення даних

Дані можна змінювати безпосередньо в таблицях. Чи доцільно, на вашу думку, застосовувати запити для виконання аналогічних операцій? Чому?

Запит на змінення даних — це запит, за допомогою якого в таблицю вносяться зміни. Можна не лише вибирати необхідні дані з таблиць, а й створювати з вибраних даних нову таблицю, змінювати дані в уже створених таблицях, додавати нові записи в створені таблиці, вилучати з таблиць записи.

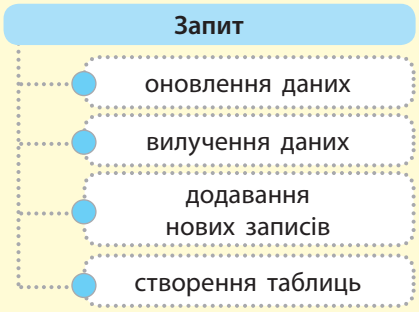


Рис. 3.15. Типи запитів на змінення

В Access існують різні типи запитів на змінення (рис. 3.15).

Порядок створення запитів на змінення в режимі конструктора такий самий, як і порядок створення звичайних запитів на вибірку даних. Потім створений запит перетворюється на запит одного з перелічених типів.

Далі розглянемо особливості розроблення запитів для створення нової таблиці та додавання записів у таблицю.

За допомогою запитів для створення нової таблиці вибираються дані з однієї або кількох таблиць і з них формується нова таблиця. Вона може бути розміщена як у поточній БД, так і в іншій, ім'я якої вказується під час створення запиту цього типу. Нова таблиця не має зв'язку з тими таблицями, з яких вона створена. Отже, якщо в таблицях-джерелах відбулися зміни, то дані в ній автоматично не оновлюються.

Розглянемо створення запиту на прикладі 1.

Приклад 1. Розробити запит, за допомогою якого на основі даних таблиць МАГАЗИНИ й КАДРИ створюється нова таблиця з іменем ДОДАТКОВА, у якій містяться поля Справа, Прізвище, Рік народження і Працівників тих магазинів, де кількість працюючих перевищує 14.

1. У відкритій БД atb виконаємо команду Створити → Макет запиту, виділимо обидві таблиці й перенесемо з таблиці КАДРИ поля Справа, Прізвище й Рік народження, а з таблиці МАГАЗИНИ — поле Працівників. Закриємо вікно Відображення таблиці. У запис Критерії поля Працівників уведемо вираз >14. Виконаємо запит і переконаємося, що отримано правильний результат.
2. Перемкнемо запит у режим конструктора й перетворимо запит на вибірку в запит на створення таблиці. Для цього на вкладці Конструктор у групі Тип запиту клацнемо кнопку Створення таблиці. Відкриється вікно Створити таблицю (рис. 3.16).

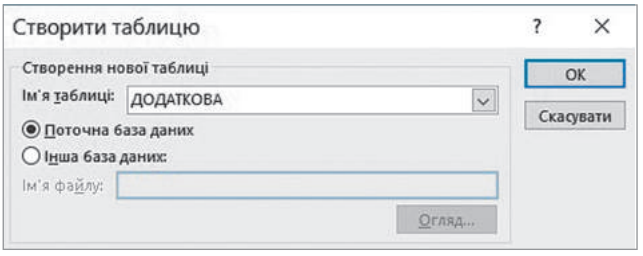


Рис. 3.16. Вікно для створення таблиці

3. У поле Ім'я таблиці введемо ім'я нової таблиці, наприклад ДОДАТКОВА, і ввімкнемо перемикач Поточна база даних, оскільки цю таблицю зберігатимемо у відкритій (поточній) БД atb. Після цього клацнемо кнопку OK.
4. На панелі швидкого доступу клацнемо кнопку Зберегти й збережемо запит із іменем Запит_7.
5. Виконаємо Запит_7 — відкриється вікно (рис. 3.17).

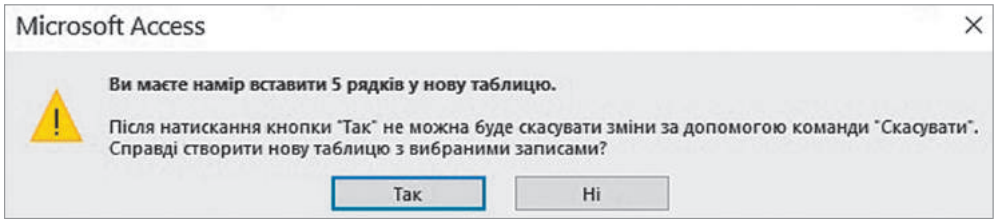


Рис. 3.17. Вікно для підтвердження збереження таблиці

6. Для збереження таблиці в поточній БД клацнемо кнопку Так. В області переходів з'явиться ім'я таблиці ДОДАТКОВА. Після цього закриємо запит і відкриємо створену таблицю, вміст якої наведено на рис. 3.18.
7. Проаналізуємо вміст таблиці й переконаємося, що вона сформована правильно.

ДОДАТКОВА

Справа	Прізвище	Рік народження	Працівників
120	Рябко Р. П.	1981	20
111	Семко М. М.	1970	20
116	Раков Г. П.	1965	20
105	Сокіл Т. Л.	1960	15
109	Шрамко Т. Л.	1961	15

Рис. 3.18. Вміст таблиці ДОДАТКОВА

Запити на додавання даних призначено для додавання нових записів у таблицю на основі опрацювання за певними критеріями даних, які вже є в раніше створених таблицях.

Інколи для додавання всіх записів усіх полів із наявної таблиці в нову доцільніше скористатися командами Копіювати і Вставити. Записи можна додавати як у відкриту, так і в закриту таблицю. Якщо записи додаються в таблицю іншої БД, то слід вказати ім'я та маршрут файла цієї БД.

Загальний порядок розроблення запитів такий:

1) створюється запит на вибірку, за допомогою якого формуються необхідні записи для додавання;
2) перетворюється запит на вибірку в запит на додавання;
3) вибирається таблиця, у яку додаватимуться записи;
4) зберігається й виконується запит.

Розглянемо порядок дій на прикладі 2.

Якщо у записі **Поле** таблиці конструктора запиту є символ «зірочка» (*), то це означає, що використовувати в запиті окремі поля цієї самої таблиці неможливо.

Приклад 2. Створити Запит_8, за допомогою якого до таблиці ДОДАТКОВА додаються з таблиць МАГАЗИНИ й КАДРИ прізвища працівників магазинів, у яких працює 13 робітників, народжених у 1975 році.

1. Створимо запит на вибірку. Із таблиці МАГАЗИНИ перенесемо в таблицю конструктора запитів поля Справа, Прізвище й Рік народження, а з таблиці КАДРИ — поле Працівників. Після виконання цього запиту має з'явитися результат, як наведено на рис. 3.19.

Запит1

Справа	Прізвище	Рік народження	Працівникіі
115	Горошко Ф. Р.	1975	13

Рис. 3.19. Запит на вибірку даних із таблиць МАГАЗИНИ й КАДРИ

2. Перетворимо створений запит на вибірку в запит на додавання. Для цього перейдемо в режим конструктора і в групі Тип запиту виконаємо команду Додавання.

3. У вікні Додавання, що відкрилося, введемо ім'я таблиці — ДОДАТКОВА, виберемо варіант Поточна база даних і клацнемо кнопку ОК.

4. Збережемо запит з ім'ям Запит_8 і виконаємо його.
У результаті відкриється вікно, як наведено на рис. 3.20.

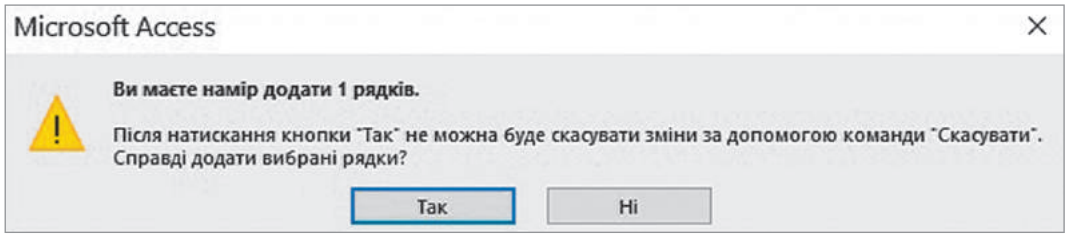


Рис. 3.20. Вікно для підтвердження додавання запису

ДОДАТКОВА

Справа	Прізвище	Рік народж	Працівникі
120	Рябко Р. П.	1981	20
111	Семко М. М.	1970	20
116	Раков Г. П.	1965	20
105	Сокіл Т. Л.	1960	15
109	Шрамко Т. Л.	1961	15
115	Горошко Ф. Р.	1975	13
115	Горошко Ф. Р.	1975	13

Рис. 3.21. Таблиця після додавання запису

5. Підтвердимо додавання запису, для чого клацнемо кнопку Так. Закриємо Запит_8 і відкриємо таблицю ДОДАТКОВА.
У результаті повинна з'явитися таблиця, як наведено на рис. 3.21.

Запитання для перевірки знань

1. Назвіть типи запитів на змінення.
2. Як перетворити запит на вибірку в запит на створення таблиці?
3. Як перетворити запит на вибірку в запит на додавання?
4. Як створити запит для створення таблиці?
5. Яка сутність запитів на додавання записів?
6. Опишіть загальний порядок створення запиту на додавання.
7. Наведіть приклад запиту для створення нової таблиці.
8. Наведіть приклад запиту на додавання.

Завдання для самостійного виконання

1. Розробіть **Запит71**, за допомогою якого на основі таблиці УЧНІ створюється таблиця ПЕРША1. Таблиця повинна містити дані про учнів 10 класу з полями **Прізвище**, **Дата народження**, **Зріст**, **Улюблений предмет**, **Клас**.
2. Створіть **Запит72**, за допомогою якого до таблиці ПЕРША1 додаються записи про учнів 9 класу з тими самими полями, що й у завданні 1. Збережіть таблицю з іменем ПЕРША2.
3. Створіть **Запит73**, за допомогою якого на основі даних таблиць КЛАСИ й УЧНІ створюється таблиця з іменем ПЕРША3 з полями **Прізвище**, **Дата народження**, **Улюблений предмет**, **Клас** для класів, у яких кількість учнів менша за 27.
4. Розробіть **Запит74**, за допомогою якого до таблиці ПЕРША3 додається поле **Інформатика**. Дані збережіть у таблиці ПЕРША4.
5. Розробіть **Запит75**, за допомогою якого до таблиці ПЕРША4 додаються записи про класи, у яких кількість учнів дорівнює 27 і які мають з історії успішність 9 балів.

4. Інтерфейс користувача. Основи мови SQL. Імпорт та експорт даних

4.1. Створення інтерфейсу користувача для введення даних у базу даних

Спробуйте самостійно визначити основні недоліки таблиць і запитів для роботи з даними з точки зору кінцевого користувача.

Як ви знаєте, дані БД зберігаються у таблицях, і в них можна безпосередньо вводити дані, редагувати їх і вилучати. Не кожен користувач має право це робити. Задля забезпечення цілісності даних і безпеки БД для більшості користувачів доступ до структури і власне таблиць БД має бути обмежений. Наприклад, продавець супермаркету має право внести дані про реалізацію конкретного товару, але не може змінити ціну товару, це право належить адміністрації.

У середовищі Access для введення даних використовується форма.

З одного боку, форми забезпечують безпеку даних від несанкціонованого доступу, а з іншого — спрощують сам процес уведення даних — користувачу не потрібно переглядати таблицю з десятками полів і відшукувати потрібні дані. Користувачу пропонується бланк, наприклад, із двома-трьома полями, у які він має право внести необхідні дані. Важливо й те, що за допомогою форми дані можуть оновлюватися одночасно у кількох пов'язаних таблицях.

Система Access 2016 має різні засоби створення форм, які містяться на вкладці Створення у групі Форми (рис. 4.1). Найпотужнішим інструментарієм створення форм є Конструктор форм.

Рис. 4.1. Засоби створення форм

Розглянемо порядок використання Конструктора форм для введення даних у БД на прикладі 1.

Із таблицею як об'єктом працює в основному адміністратор БД. Для роботи з даними користувачам потрібен більш простий і зручний інтерфейс. Для кожної категорії працівників бажано розробити свої правила й засоби роботи з БД.

Форма — це фактично електронний бланк, який для кожної групи користувачів має свою структуру. У цій структурі містяться імена полів, доступні саме для цієї групи. Після заповнення полів дані потрапляють у відповідні поля таблиць для подальшого збереження.

Приклад 1. Створити для БД atb на основі таблиці МАГАЗИНИ форму з іменем Форма_6 і з полями Номер магазину, Адреса, Директор, Телефон. Перед першими двома полями ввести текст Координати, а ще перед двома — Контакти.

1. Виділимо в області переходів таблицю МАГАЗИНИ, активуємо вкладку Створити й у групі Форми виконаємо команду Конструктор форм. На екрані з'явиться порожня форма з розділом Подробиці й буде активовано вкладку Конструктор.

2. У групі Колонтитули клацнемо кнопку Назва. У результаті відкриється вікно (рис. 4.2), у якому містяться підрозділи Верхній колонтитул форми, Нижній колонтитул форми і текстове поле з іменем Форма1.
3. Установимо курсор у поле Форма1, введемо текст заголовка, наприклад Магазини нашого району, і клацнемо кнопку Enter.
4. У верхню частину області Подробиці уведемо текст Координати, який відповідно до умови має міститися перед першими двома полями. Для цього клацнемо кнопку Напис, установимо курсор у потрібну позицію області Подробиці, уведемо текст Координати й клацнемо кнопку Enter.
5. У правій частині вікна має розташовуватися поле Список полів (якщо воно відсутнє, клацнемо кнопку Додавання наявних полів, і воно з'явиться). У цьому полі клацнемо кнопку Відображати всі таблиці. Ліворуч від назви таблиці МАГАЗИНИ встановимо перемикач у положення (–) — під назвою таблиці відкриється список її полів. Перемістимо поле Номер магазину в те місце розділу Подробиці, куди слід помістити це поле. У результаті на формі буде розташовано два пов'язані елементи: власне поле введення і підпис для нього (підпис співпадає з іменем цього поля).
6. Аналогічно перемістимо до бланку форми поле Адреса.
7. Після двох перших полів розмістимо назву ще двох полів. Для цього за аналогією у групі Елементи керування клацнемо кнопку Напис, установимо курсор у потрібну позицію області Подробиці, уведемо назву Контакти й натиснемо клавішу Enter. Перемістимо поля Директор і Телефон у розділ Подробиці. Розмістимо елементи форми більш зручно (рис. 4.3).

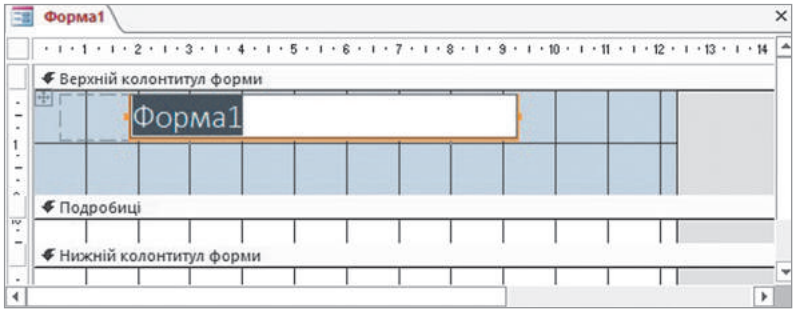


Рис. 4.2. Порожня форма в режимі конструктора

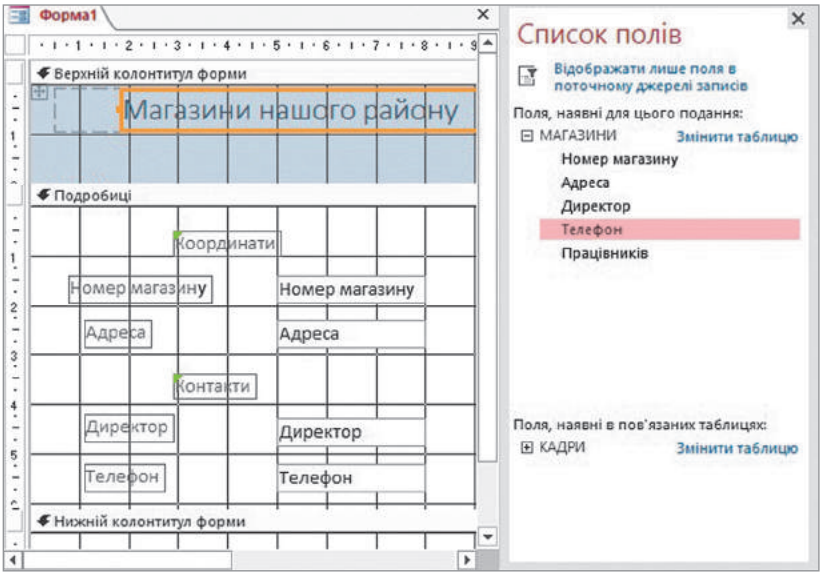


Рис. 4.3. Форма, створена в режимі конструктора

Після цього можна закрити вікно Список полів. Якщо у форму необхідно додати поля з кількох таблиць, то їх потрібно розкрити так само, як і поля таблиці МАГАЗИНИ.

8. Збережемо форму з іменем Форма_6. Тепер форму можна переглянути в таких режимах: Режим форм, Подання таблиці, Режим розмітки.

На рис. 4.4 створену форму відкрито в режимі форми.

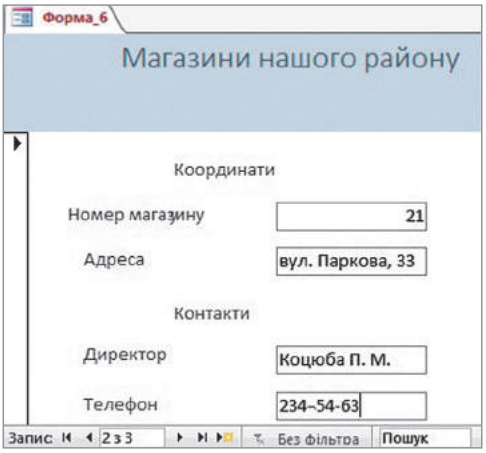


Рис. 4.4. Форму відкрито в режимі форми

Зовнішній вигляд форми можна налаштувати з урахуванням потреб користувача. Для цього слід відкрити контекстне меню форми: установити курсор на вільному місці форми, відкритої в режимі конструктора, і клацнути праву кнопку миші.

Вміст контекстного меню форми зображено на рис. 4.5.

Щоб змінити колір фону форми, вказівник миші слід установити на команді Колір заливки/фону та в наборі кольорів, що відкриється, вибрати потрібний колір.

Або вибрати інший спосіб: відкрити контекстне меню заголовка форми чи інших її компонентів і встановити потрібні кольори. На формі можна також установити або зняти лінійку, сітку чи виконати інші налаштування.

Один із варіантів налаштування зовнішнього вигляду форми наведено на рис. 4.6.

За допомогою кнопок навігації можна перейти до будь-якого запису.

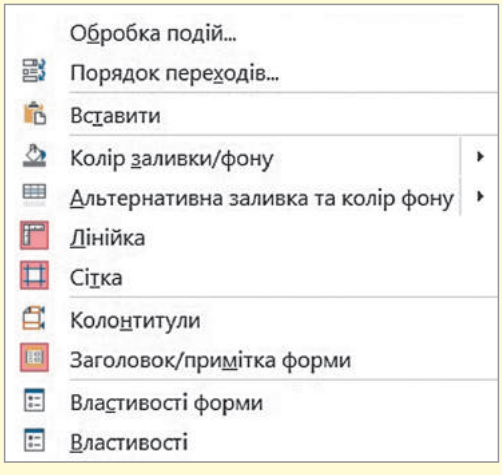


Рис. 4.5. Контекстне меню форми

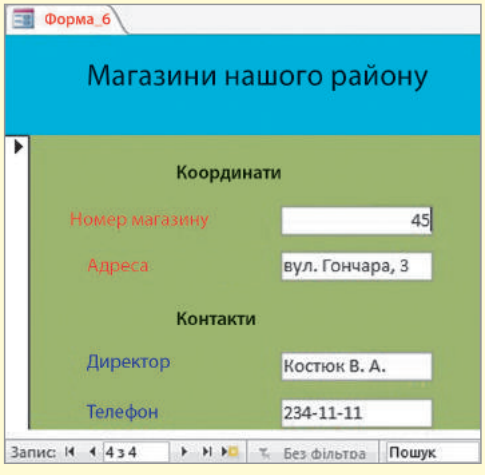


Рис. 4.6. Варіант оформлення зовнішнього вигляду форми

Порядок додавання записів на основі створеної форми розглянемо на прикладі 2.

Приклад 2.

1. Відкриємо Форму_6 у режимі форми. Перейдемо на порожній запис, для чого клацнемо, наприклад, кнопку Створити запис, розташовану на панелі навігації.
2. Уведемо в порожній рядок форми дані, наприклад такі: Номер магазину — 6; Адреса — вул. Річкова, 24; Директор — Середа К. М.; Телефон — 234-67-92 (рис. 4.7).
3. Збережемо й закриємо форму. Таблиця МАГАЗИНИ, відкрита в режимі подання таблиці, набуде вмісту, як наведено на рис. 4.8.

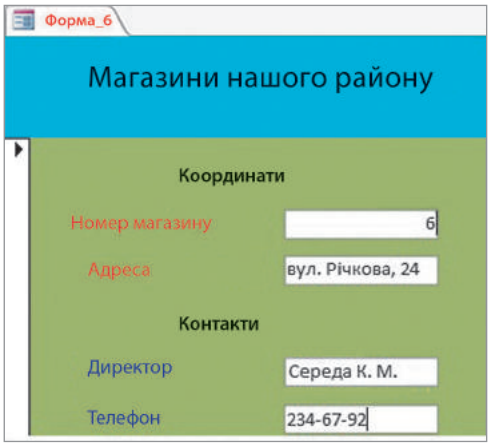


Рис. 4.7. Запис, доданий у таблицю

МАГАЗИНИ

Номер магазину	Адреса	Директор	Телефон	Працівникі	Клацніть, щоб додати
6	вул. Річкова, 24	Середа К. М.	234-67-92	15	
21	вул. Паркова, 33	Коцюба П. М.	234=54-63	20	
31	вул. Печерська, 21	Борзов А. С.	234-50-50	13	
45	вул. Гончара, 3	Костюк В. А.	234-11-11	0	
0				0	

Рис. 4.8. Вміст таблиці МАГАЗИНИ після додавання запису

Зверніть увагу на те, що поле Працівників для введеного запису містить нуль, оскільки у Формі_6 такого поля створено не було і дані в нього не вводилися.

Запитання для перевірки знань

1. Яке призначення **Конструктора форм**?
2. Поясніть порядок створення форм за допомогою **Конструктора форм**.
3. Як перейти до будь-якого запису?
4. Поясніть порядок оформлення загального вигляду форми.

Завдання для самостійного виконання

1. Створіть за допомогою конструктора форм для таблиці КЛАСИ форму з полями **Клас**, **Учнів**, **Класний керівник**. Здійсніть навігацію по записах форми.
2. Створіть за допомогою конструктора форм для таблиці УЧНІ форму з полями **Прізвище**, **Клас**, **Інформатика**.
3. Створіть за допомогою конструктора форм для таблиці КЛАСИ та таблиці УЧНІ форму з полями **Клас**, **Класний керівник**, **Прізвище**, **Адреса**, **Дата народження**. Перегляньте форму в різних режимах. Самостійно виберіть колір кожного компонента форми.

4.2. Основи мови запитів SQL

Поміркуйте, як мова запитів SQL розширює функціональні можливості СУБД для роботи з даними.

Сучасні системи управління реляційними БД, у тому числі Access 2016, містять потужні візуальні засоби, зрозумілий і зручний графічний інтерфейс, що забезпечує ефективну роботу з БД. Водночас існують спеціальні мови для створення й супроводу БД. Однією з таких мов є SQL (Structured Query Language — структурована мова запитів).

У системі Access 2016 реалізовано частину SQL, що забезпечує формування запитів і роботу з ними. Далі стисло розглянемо основні відомості саме про мову запитів SQL.

SQL фактично стала стандартом мови реляційних БД. Вона постійно розвивається й удосконалюється, змінюються її міжнародні стандарти.

Як і інші мови програмування, SQL має власний синтаксис. У синтаксичних конструкціях використовуються такі поняття, як ключові слова, оператори, інструкції, речення та ін.

У мові запитів SQL найчастіше використовуються такі оператори:

- SELECT — визначає поля, із яких необхідно вибрати дані;
- FROM — визначає таблицю, поля якої вказано в реченні SELECT; ключові слова SELECT і FROM завжди використовуються разом;
- WHERE — визначає умову відбору полів, за якою вибираються дані;
- ORDER BY — визначає порядок сортування отриманих результатів;
- GROUP BY — визначає порядок групування записів.

Інструкція — це логічно завершена конструкція, яка може інтерпретуватися самостійно. Вона складається із речень і закінчується крапкою з комою.

Інструкції можуть містити коментарі, які не впливають на їх виконання. Найчастіше застосовуються однорядкові коментарі, які починаються двома символами «--».

Речення — це частина інструкції, що обов'язково містить ключове слово, яке визначає його назву.

Наприклад, SELECT Прізвище називають реченням SELECT, а WHERE Посада = 'вчитель' — реченням WHERE.

У мові SQL структура найуживаніших речень така:

```
SELECT <список полів>              -- речення SELECT
FROM <ім'я таблиці>                -- речення FROM
WHERE <ім'я поля> = <умова>;       -- речення WHERE
```

Наприклад, за допомогою інструкції:

```
SELECT Прізвище, Адреса, Телефон
FROM Школа
WHERE Посада = 'вчитель';
```

із таблиці Школа вибираються всі записи, у полі Посада яких є значення вчитель. Результуючий набір записів містить поля Прізвище, Адреса, Телефон.

Конструкції у фігурних дужках є обов'язковими, у квадратних дужках — необов'язковими.

Багато термінів, понять, синтаксичних правил і властивостей мови SQL збігаються з відповідними назвами класичних мов програмування. Усі конструкції мови (ключові слова, оператори тощо) однаково сприймаються і великими, і малими літерами.

За стандартом мови SQL для імен об'єктів (таблиць, полів та ін.) використовується латинський алфавіт, але в багатьох випадках допускається використання й національного алфавіту.

Розглянемо порядок створення найпростішого запиту за допомогою мови SQL на прикладі.

Приклад. Створити найпростіший запит на основі таблиці КАДРИ за допомогою мови запитів SQL.

1. У системі Access 2016 виконаємо команду Створити → Макет запиту.
2. Закриємо вікно Відображення таблиці.
3. На вкладці Конструктор у групі Результати клацнемо кнопку SQL. Відкриється вікно Запит1, у робочому полі якого висвітлиться оператор SELECT. Він обов'язково використовується з оператором FROM і має таку мінімальну загальну структуру:

SELECT <список імен полів>
FROM <ім'я таблиці>;
Наприклад, інструкція:
SELECT*
FROM <ім'я таблиці>;
забезпечує виведення всіх полів таблиці.

Після виконання інструкції:
SELECT Справа, Прізвище, Освіта, Стаж
FROM КАДРИ;
отримаємо результат, як наведено на рис. 4.9.

Запит1

Справа	Прізвище	Освіта	Стаж
105	Сокіл Т. Л.	середня	27
109	Шрамко Т. Л.	середня	24
120	Рябко Р. П.	вища	8
111	Семко М. М.	середня	16
116	Раков Г. П.	вища	19
132	Таран В. Д.	вища	15
115	Горошко Ф.Р.	середня спеціальна	17
0			0

Рис. 4.9. Записи таблиці КАДРИ з окремими полями

Як бачимо, виводяться записи з полями, зазначені в операторі SELECT.

Зауважимо, що порядок розміщення полів у реченні SELECT може бути довільним і не збігатися з порядком їх розміщення в початковій таблиці.

Поля виводяться в порядку їх розміщення у реченні SELECT.

Запитання для перевірки знань

1. Які оператори містить найпростіша інструкція мовою SQL?
2. Поясніть загальний формат найпростішої інструкції мовою SQL.
3. Наведіть приклад найпростішої інструкції мовою SQL.
4. Які дужки використовують для обов'язкових конструкцій?

4.3. Імпорт і експорт об'єктів баз даних

Чому, на вашу думку, виникає потреба імпортувати й експортувати дані із БД в інші об'єкти?

На практиці досить часто доводиться одночасно працювати не з одним файлом БД, а з кількома, у тому числі розміщеними в Інтернеті. У таких випадках виникає потреба в обміні даними (об'єктами) між БД, створеними за допомогою однієї СУБД (наприклад, Access 2016), різних СУБД (наприклад, Access і Paradox).

Окрім того, інколи буває потрібно обмінятися даними між Access та іншими програмами пакета Microsoft Office Windows, наприклад Word-, Excel-, Outlook-, HTML- і XML-документами.

Найпростішим способом обміну об'єктами між документами є копіювання об'єктів і вставлення за допомогою буфера обміну. Наприклад, у документ, створений у текстовому процесорі Word, можна додати рисунок, створений у графічному редакторі Paint.

У системі Access 2016 для обміну об'єктами здійснюють спеціальні операції, що отримали назву імпорту й експорту. Крім того, для отримання даних із інших зовнішніх джерел системою Access підтримується технологія зв'язування, яка тут не розглядається.

У процесі імпортування об'єкти перетворюються у формат Access 2016 і розміщуються в новому об'єкті, а в об'єкті-джерелі залишаються без змін. Наприклад, нова і початкова таблиці існують незалежно одна від одної. Технології імпорту з різних систем (Word, Excel та ін.) відрізняються одна від одної.

Загальний порядок імпортування однаковий для всіх систем.

Етап 1	Відкриття БД-приймача і вибір системи, з якої здійснюватиметься імпортування (наприклад, Access, Excel).
Етап 2	Вибір об'єктів, які необхідно імпортувати, і налаштування параметрів їх імпортування (у разі необхідності).
Етап 3	Виконання імпортування.

За своїм змістом експорт є операцією, зворотною до імпорту. У процесі експортування здійснюється перенесення даних із об'єктів системи Access в іншу БД цієї системи, а також у зовнішні файли різних форматів. Під час експортування дані перетворюються у новий формат, а об'єкти-джерела залишаються незмінними.

Оскільки загальні принципи імпортування й експортування всіх типів об'єктів БД ідентичні, далі розглядатимемо лише технологію та правила імпортування об'єктів із однієї БД Access 2016 в іншу БД Access 2016.

Для отримання об'єктів у БД із зовнішніх джерел в Access 2016 використовується **технологія імпорту**, а для передавання об'єктів в інші застосунки — **технологія експорту**.

Імпортувати можна будь-які об'єкти БД. Одночасно можна імпортувати як кілька об'єктів одного типу (наприклад, кілька запитів), так і кілька об'єктів різного типу (наприклад, кілька таблиць і запитів).

Джуді Маллой — американська письменниця та програмістка-самоучка. Винайшла власну систему БД для своїх романів. У 1986 році написала й запрограмувала гіпертекстовий роман «Дядько Роджер» («Uncle Roger»). Це перший онлайн-проект з посиланнями, які змінюють сюжет залежно від уподобань читача.

Розглянемо технологію імпорту на прикладі.

Приклад. Здійснити імпорт таблиці КАДРИ і запиту Запит_1 із БД atb у БД skola.

Отже, джерелом є база atb, а приймачем — база skola.

1. Відкриємо БД-приймач skola, активуємо вкладку Зовнішні дані. Панель інструментів набуде вигляду, як зображено на рис. 4.10.

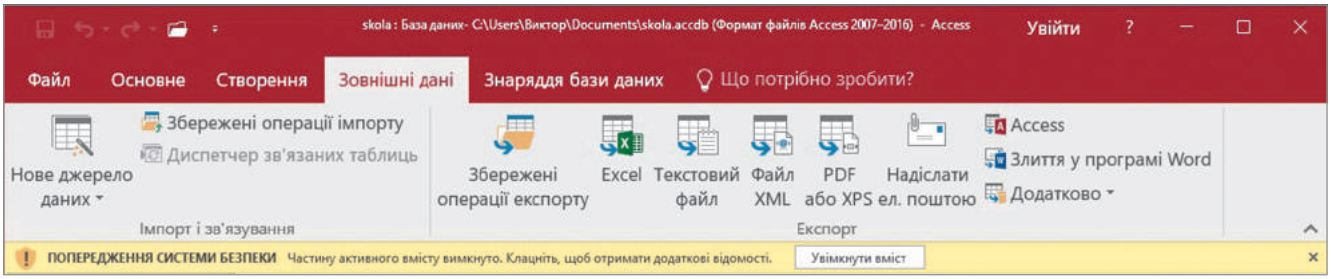


Рис. 4.10. Панель інструментів вкладки **Зовнішні дані**

2. У групі Імпорт і зв'язування відкриємо меню кнопки Нове джерело даних, установимо в ньому курсор на назві Із бази даних. Відкриється перелік джерел, як наведено на рис. 4.11.
3. Оскільки виконується імпорт об'єктів із однієї БД Access в іншу БД Access, клацнемо кнопку Access. Відкриється перше вікно Майстра імпорту, як наведено на рис. 4.12.

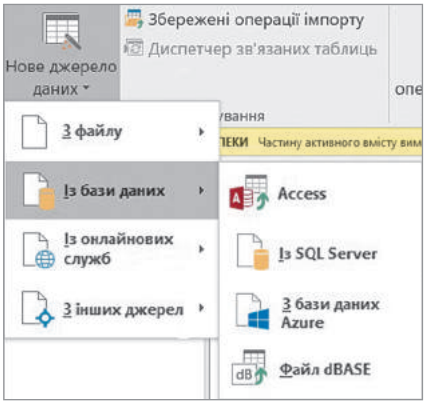


Рис. 4.11. Перелік джерел даних для імпортування

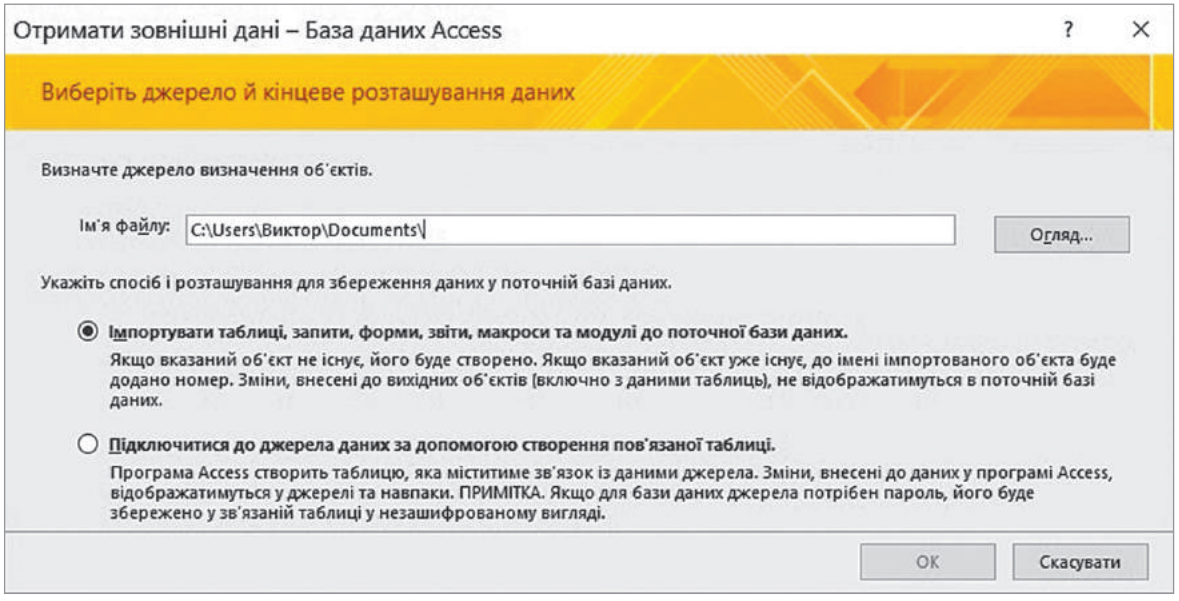


Рис. 4.12. Перше вікно **Майстра імпорту**

У поле Ім'я файлу цього вікна можна ввести повне ім'я файла БД-джерела. Можна також скористатися кнопкою Огляд... і вибрати файл atb. Клацнемо кнопку Огляд та знайдемо ім'я atb, увімкнемо перемикач Імпортувати таблиці, запити... і клацнемо кнопку ОК. Відкриється вікно Імпортувати об'єкти (рис. 4.13).

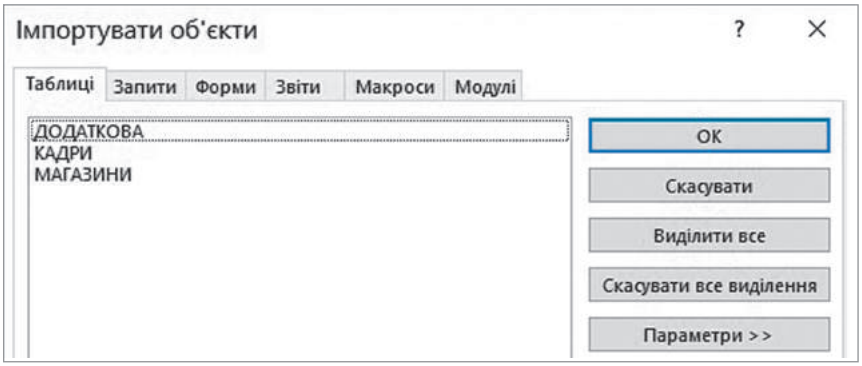


Рис. 4.13. Вікно для імпортування об'єктів

У цьому вікні містяться вкладки об'єктів БД-джерела atb. На вкладці Таблиці виберемо таблицю КАДРИ, для чого встановимо курсор на її імені й клацнемо праву кнопку миші. Потім на вкладці Запити виберемо Запит_1.

4. Можна налаштувати деякі параметри імпортування вибраних об'єктів. Для цього клацнемо кнопку Параметри>>. У результаті у вікні відобразиться область налаштування параметрів імпорту (рис. 4.14).

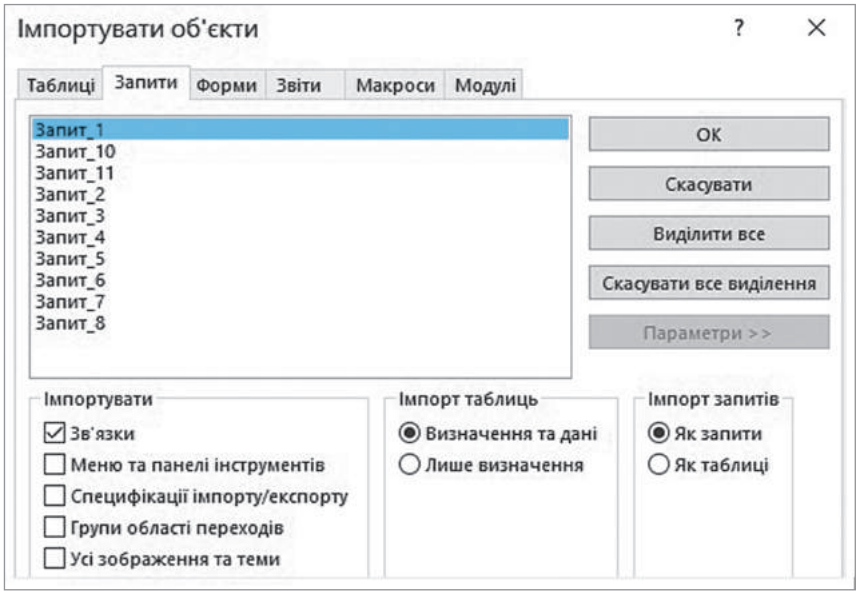


Рис. 4.14. Вікно з областю налаштування параметрів імпорту

Розглянемо призначення параметрів групи Імпорт таблиць.

Перемикач Визначення та дані встановлюється в тому випадку, коли з БД-джерела необхідно імпортувати і структури, і дані всіх таблиць, вибраних користувачем у допоміжному вікні.

Якщо вибрати перемикач Лише визначення, то імпортуватимуться тільки структури таблиць, а не дані. Увімкнемо перший перемикач.

Запит_1 створено на основі таблиці КАДРИ, яку вже імпортовано. Отже, для його запуску в БД skola жодних операцій виконувати не треба, увімкнемо перемикач Як запити.

5. Для збереження всіх налаштувань клацнемо кнопку OK. Одразу почнеться імпортування вибраних об'єктів у БД skola. Після завершення імпортування на екран буде виведено повідомлення про результат виконання операції, як наведено на рис. 4.15.
6. Усі кроки, що виконувалися під час імпортування, можна зберегти для того, щоб за потреби можна було повторити їх без використання Майстра імпорту (за замовчуванням ці кроки не зберігаються). Для збереження виконаних кроків у вікні, що відкрито, необхідно ввімкнути прапорець Зберегти етапи імпортування. Але тут виконувати цю операцію немає потреби, тому просто клацнемо кнопку Закрити. Якщо тепер відкрити БД skola, то в ній побачимо таблицю КАДРИ і Запит_1. Їх вміст такий самий, як у БД atb.

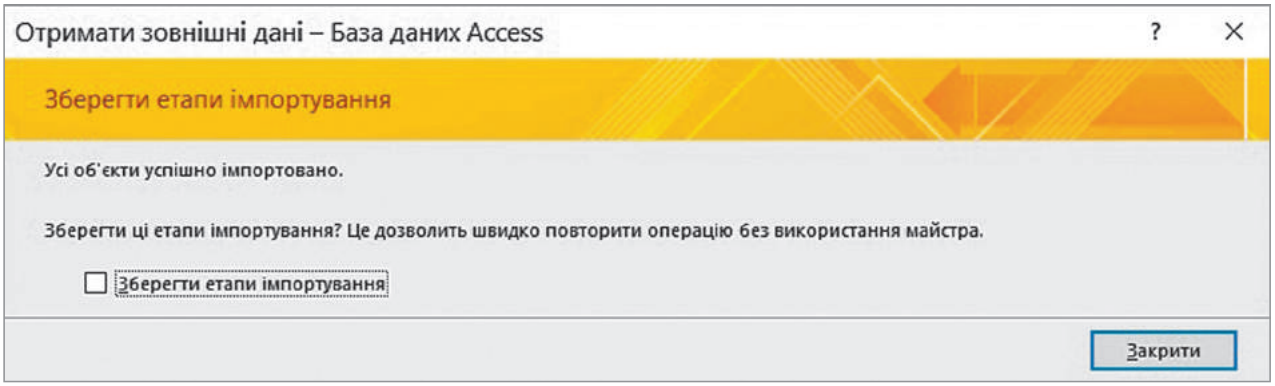


Рис. 4.15. Вікно для збереження етапів імпортування

Для імпортування таблиці із книги Excel потрібно відкрити БД-приймач, на вкладці **Зовнішні дані** клацнути кнопку **Excel** і вибрати команду **Майстра імпорту**.

Щоб імпортувати текстовий файл, його необхідно попередньо структурувати так, щоб кожний рядок файла був записом, а кожний запис поділявся на окремі поля.

Для розмежування полів часто використовується крапка з комою. Приклад структурованого текстового файла:

Микола; 050-400-22-33; 21 січня
Олена; 063-333-11-55; 5 травня

Запитання для перевірки знань

1. Опишіть найпростіший спосіб обміну даними між програмами Windows.
2. Між якими програмами можна здійснювати обмін даними з БД Access 2016?
3. Яка сутність імпорту даних у БД Access?
4. З якою метою здійснюється налаштування параметрів імпортування?
5. Поясніть загальний порядок імпортування даних у БД Access.
6. Що використовують для розмежування полів?

Завдання для самостійного виконання

Для виконання завдання створіть на жорсткому диску БД **persha**. Імпортуйте в БД **persha** форму з іменем **Форма_1** із БД atb. Перевірте правильність імпортування.

Тест 1 Тест 2 Тест 3 Тест 4

Тестові завдання з автоматичною перевіркою результату на сайті interactive.ranok.com.ua

Розділ 2. АЛГОРИТМИ

5. Алгоритми і числа

5.1. Методи проектування і подання алгоритмів

Пригадайте, у якій формі ви подавали алгоритми, яких правил дотримувались у процесі розроблення програм.

Проектування алгоритмів і програм є виключно творчим процесом. Не існує універсального методу розроблення алгоритму розв'язування для будь-якого завдання. Для кожного завдання необхідно знайти свій, найбільш раціональний метод.

У процесі проектування алгоритму намагаються:

- забезпечити мінімальний час розв'язування задачі;
- використати мінімальний обсяг пам'яті;
- досягти потрібної точності й надійності обчислення;
- забезпечити ефективне використання можливостей наявних бібліотек, зокрема мінімізувати вартість розроблення алгоритму.

Методи проектування алгоритмів класифікуються за багатьма ознаками. Основними з них є ступінь автоматизації проектування алгоритмів і програм та методологія проектування програмних продуктів (рис. 5.1).

Неавтоматизовані методи використовуються у процесі розроблення невеликих і нескладних програмних продуктів за участю невеликої кількості розробників. Такі методи застосовуються, зокрема, у процесі розроблення програмних продуктів навчального призначення.

Автоматизовані методи застосовуються у великих компаніях і потребують додаткового апаратно-програмного забезпечення і високої кваліфікації працівників.

В основі *структурного проектування програмних продуктів* лежать послідовна декомпозиція і структурування програмного продукту на окремі складові. *Структурне проектування програмних продуктів* засноване на створенні алгоритмів із базових структурних алгоритмічних одиниць. Доведено, що такими одиницями є: слідування, розгалуження і повторення (цикли). Ці алгоритмічні структури послідовно з'єднуються або укладаються одна в одну з дотриманням певних правил. Алгоритм виконується послідовно зверху вниз.

Правильність виконання алгоритму можна відслідковувати на кожному етапі його побудови і виконання. Будь-який алгоритм може бути еквівалентно поданий структурованим алгоритмом, що складається з базових алгоритмічних структур.

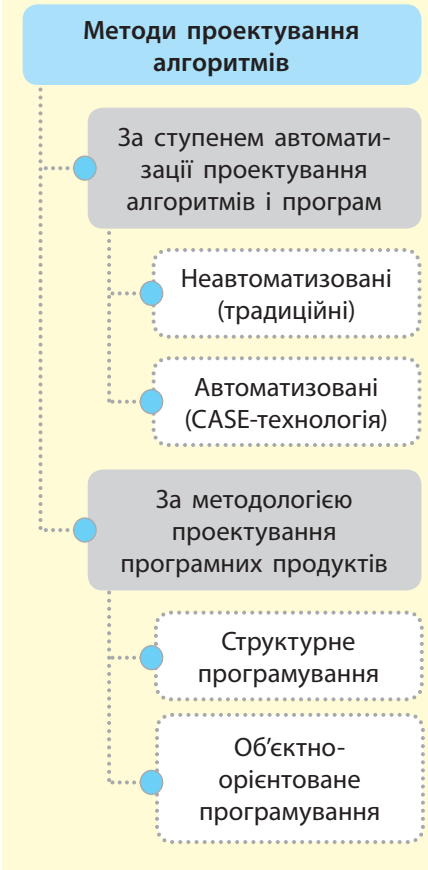


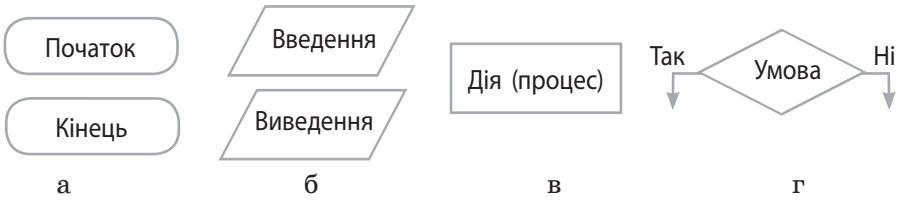
Рис. 5.1. Класифікація методів проектування алгоритмів

Розглянемо типові методи структурного проектування програмних продуктів:

Метод	Опис
Спадний (згори вниз)	Його сутність полягає в тому, що задача поступово (за кроками) ділиться на ряд допоміжних підзадач (підалгоритмів), кожна з яких може бути реалізована сукупністю простих і елементарних операцій (процедур).
Висхідний (знизу догори)	Вже наявні й заздалегідь розроблені допоміжні алгоритми розв'язування окремих підзадач поступово об'єднуються в загальну структуру доти, доки не буде досягнуто розв'язання поставленого завдання.
Модульний	Модуль — це окрема самостійна частина алгоритму (деякий блок), що має свою назву, функціональну цілісність і завершеність. Посилання на модуль здійснюється за допомогою його імені. Виклик і актуалізація модуля можливі лише через його заголовок. Перевага модульного методу полягає в тому, що різні модулі одночасно можуть розробляти різні фахівці. Кожний модуль може тестуватися і налагоджуватися окремо від інших.

Згадаємо способи подання алгоритмів:

Спосіб	Опис
Словесний	Передбачає опис алгоритму природною мовою, широко застосовується у повсякденному житті (наприклад, у вигляді інструкцій з експлуатації приладів, рецептів виготовлення ліків тощо). Інструкція складається з указівок, форма запису яких довільна. Головне, щоб указівки були точними й зрозумілими всім користувачам. Словесний спосіб є досить простим і доступним, проте опис алгоритмів часто є громіздким, а їхні вказівки можуть сприйматися різними виконавцями неоднозначно.
Словесно-формульний	Використовує природну мову, а також спеціальні символи (наприклад, хімічні формули, математичні вирази та ін.), що застосовуються в певній науковій галузі. Наприклад, розв'яжемо квадратне рівняння $ax^2+bx+c=0$. 1. Уведемо значення змінних a, b, c. 2. Обчислимо дискримінант $d = b^2-4ac$. 3. Якщо $d<0$, то виконаємо пункт 6, інакше — пункт 4. 4. Знайдемо корені рівняння: $x_1=\frac{-b-\sqrt{d}}{2a}$; $x_2=\frac{-b+\sqrt{d}}{2a}$. 5. Перейдемо до пункту 7. 6. Дискримінант від'ємний. Рівняння не має розв'язків. 7. Кінець.
Графічний	Передбачає подання алгоритму у вигляді геометричних фігур (блоків), з'єднаних стрілками (лініями зв'язку). Подання алгоритмів за допомогою блоків називають блок-схемами, вони мають високу наочність.

	На рис. 5.2 зображено основні блоки, за допомогою яких створюються блок-схеми алгоритмів.  Рис. 5.2. Основні графічні позначення на блок-схемах: термінатори (*а*); дані (*б*); дія (процес) (*в*); перевірка умови (*г*)
Комбінований	Поєднує в собі подання алгоритмів, засноване на словесному, словесно-формульному і графічному методах.

Об'єктно-орієнтоване проектування (ООП) програмних продуктів засноване на тому, що кожний об'єкт об'єднує дані й програми (методи) їх опрацювання в єдину конструкцію. Кожний об'єкт належить до відповідного класу. При цьому створюється ієрархія класів, визначаються властивості об'єктів і розробляються методи їх опрацювання, а також дотримуються специфічні принципи ООП.

Запитання для перевірки знань

1. Які існують методи подання алгоритмів?
2. За якими ознаками класифікують методи проектування програмних продуктів?
3. Назвіть методи структурного проектування програмних продуктів.
4. Поясніть сутність структурного проектування.
5. У чому полягає сутність об'єктно-орієнтованого програмування?
6. Які критерії слід ураховувати під час проектування алгоритму?
7. Назвіть переваги й недоліки кожного методу проектування алгоритмів.

Завдання для самостійного виконання

1. Користуючись словесним способом подання, опишіть алгоритм виправлення помилок у слові **алгарітм** у текстовому процесорі Word, щоб отримати слово **алгоритм**.
2. Відомо, які оцінки з усіх предметів отримали за І півріччя учні Ніна та Антон. Розробіть алгоритм визначення, у кого з учнів середній бал успішності вищий, у графічний спосіб.
3. Знайдіть в Інтернеті відомості про відстані від Києва до трьох районних центрів вашої області. Користуючись словесно-формульним способом, розробіть алгоритм визначення, який із них розташований ближче до Києва.
4. Знайдіть в Інтернеті назви п'яти найбільших за кількістю населення міст вашої області. Користуючись словесно-формульним способом, розробіть алгоритм визначення назви міста, в якому мешкає людей найбільше.
5. Розробіть блок-схему алгоритму обчислення виразу $y=\begin{cases}\left(a^3+b^2\right)\cdot c-1, & \text{якщо } a=0,\\ a^3+b^2, & \text{якщо } a\neq 0.\end{cases}$
6. Розробіть блок-схему алгоритму обчислення виразу $y=\begin{cases}a^2-bx, & \text{якщо } x>0,\\ a^2-\left(c+\dfrac{x}{b}\right), & \text{якщо } x\leqslant 0.\end{cases}$
7. У банк покладено S1 грн під k відсотків річних. Розробіть блок-схему алгоритму визначення, через скільки років сума вкладу перевищуватиме S2 грн.

5.2. Поняття про кодування і складність алгоритмів

З якими мовами програмування ви вже знайомі? Чи можна один і той самий алгоритм реалізувати мовою програмування різними способами?

Для того самого алгоритму можна розробити різні варіанти програм, які відрізняються наочністю, обсягом потрібної пам'яті, швидкістю виконання, формою подання отриманих результатів та ін.

Кодування алгоритму — це запис алгоритму мовою програмування.

У програмі слід використовувати раціональну довжину рядків, щоб її можна було легко модифікувати. Не бажано застосовувати довгі рядки.

Структурне програмування в Python виконується за допомогою обов'язкових відступів. Складені оператори об'єднують у блоки. Це означає, що відступи інструкцій кожного блоку встановлюються автоматично, але після останньої інструкції блоку програміст має змінити відступ самостійно.

За наданим програмним кодом легко встановити умову задачі. Приклад програмного коду мовою Python зображено на рис. 5.3.

```
kom = int(input ("Увести номер команди:"))
if (kom == 1 or kom == 2 or kom == 5):                    # Команда Іспанії
    kr = "Іспанія"
else:
    if (kom == 3 or kom == 7):                            # Команда Німеччини
        kr = "Німеччина"
    else:
        if (kom == 4 or kom == 9 or kom == 10):           # Команда Великобританії
            kr = "Великобританія"
        else:
            if (kom == 6 or kom == 8):                    # Команда Португалії
                kr = "Португалія"
            else:
                kr = "Рейтинг невідомий"
print ("", kr)
input ()
```

Рис. 5.3. Приклад структурованого тексту програми

Така програма є досить наочною, її легко читати, розуміти та здійснювати пошук помилок. Простим і водночас потужним засобом забезпечення наочності програми є використання коментарів.

У програмному коді доцільно використовувати не однобуквені ідентифікатори, а смислові назви, так, масив краще позначити не буквою m, а ідентифікатором masyv або іншим.

Наведемо деякі рекомендації для підвищення ефективності програм.

1. Фрагменти коду, що багаторазово повторюються, бажано оформляти у вигляді функції. Це дозволяє скоротити довжину програми, а відповідно й обсяг потрібної пам'яті. Використання функцій не тільки скорочує довжину програми, а й спрощує процес її розроблення і налагодження, підвищує структурованість і надійність програм.
2. З метою скорочення кількості операцій у складному математичному виразі доцільно виділяти та окремо обчислювати прості вирази.

   Пояснимо сутність прийому на прикладі. Припустимо, що потрібно обчислити вираз: y := (a+c+b–d)/(2/(a+c)–3)*(b–d)*(a+c).

   Щоб обчислити вираз, потрібно виконати 4 операції додавання, 3 операції віднімання, 2 операції множення, 2 операції ділення. Обчислимо вираз за схемою:

   x1 := a+c; x2 := b–d;

   y := (x1+x2)/(2/x1–3)*x2*x1.

   Щоб обчислити вираз, потрібно виконати: 2 операції додавання, 2 операції віднімання, 2 операції множення і 2 операції ділення.

   Як бачимо, кількість арифметичних операцій скоротилася на три.
3. Вирази, які не змінюються в тілі циклу, доцільно для унаочнення програми виконувати до початку тіла циклу.

Складність алгоритму — це функція, що дозволяє визначити, як швидко збільшується час роботи алгоритму зі збільшенням обсягу даних.

Складність алгоритму оцінюється людськими ресурсами, потрібними для розробки та тестування алгоритму, й обчислювальними ресурсами.

Із практичної точки зору обчислювальні ресурси оцінюються нині двома основними критеріями: обсягом пам'яті, потрібної для реалізації алгоритму, та кількістю часу, необхідного для виконання алгоритму.

Для оцінки складності алгоритмів існує багато критеріїв. Найбільшу увагу приділяють порядку росту необхідних для розв'язання задачі часу та розміру пам'яті при збільшенні обсягу вхідних даних.

Відзначимо, що ця кількість реального часу є неоднозначною оцінкою, оскільки залежить від параметрів апаратних засобів комп'ютера. На одному комп'ютері алгоритм може бути виконаний за один проміжок часу, а на іншому — за інший.

Час виконання алгоритму безпосередньо залежить від вхідних даних: їх якості та кількості.

Запитання для перевірки знань

1. Як досягається наочність програмного продукту?
2. Що слід розуміти під складністю алгоритму?
3. Як реалізується структуризація програмного коду в мові Python?
4. За якими основними критеріями оцінюються обчислювальні ресурси?
5. Як краще позначати масив?
6. Назвіть основні рекомендації для підвищення ефективності програми.

5.3. Основні поняття теорії чисел

5.3.1. Системи числення

Якими системами числення ви користувалися? Чому, на вашу думку, дані в комп'ютері подаються у двійковій системі числення?

Система числення — сукупність правил запису (зображення) чисел за допомогою символів (цифрових знаків) і виконання операцій над ними.

Розрізняють непозиційні й позиційні системи числення.

У непозиційних системах числення кількісне значення цифри не залежить від її місця розташування в зображенні числа. Такі системи складні для запису чисел і виконання над ними арифметичних операцій, тому вони сьогодні майже не застосовуються (приклад 1).

У позиційних системах числення кількісне значення цифри залежить не лише від значення самої цифри, а й від її місця (позиції) у записі числа (приклад 2).

Кожну позицію цифри в числі називають розрядом. Зазвичай для цілих чисел використовується така нумерація розрядів: молодший розряд цілого числа має номер нуль, а кожний наступний номер збільшується на одиницю. Для дійсних чисел старший розряд у дробовій частині має номер -1, а кожний наступний номер дробового разряду зменшується на одиницю.

Таким чином, якщо ціла частина числа має n розрядів, а дробова — m розрядів, то старший розряд цілої частини має номер $n-1$, а молодший розряд дробової частини — номер $-m$.

Основними характеристиками позиційних систем числення є основа системи числення, вага розрядів, значення цифр, які використовуються в системі числення.

Основою системи числення (q) зазвичай називають кількість цифр, які можуть використовуватися в записі числа. Найчастіше для цього використовуються числа натурального ряду, включаючи нуль (приклад 3).

Вага розрядів у позиційних системах числення найчастіше дорівнює основі системи числення у степені номера розряду (приклад 4).

Десяткова система є прикладом системи з природною вагою розрядів, тобто такою, у якій у цілому числі вага кожного наступного розряду більша від ваги попереднього розряду в кількість разів, що дорівнює основі системи числення, а в дробовій — менше на таке саме значення.

На практиці застосовуються і системи числення з вагою розрядів, відмінною від природної ваги. Таку вагу розрядів називають *штучною*. Системи числення зі штучною вагою розрядів застосовують у спеціальних цілях, наприклад для захисту від перешкод.

Приклад 1. У наші дні з непозиційних систем числення збереглася римська система, у якій числа записуються за допомогою цифр: I (один), V (п'ять), X (десять), L (п'ятдесят), C (сто) і т. д.

Приклад 2. У числі 64 кількісне значення цифри 6 дорівнює 60, а у числі 40,6 — тільки 0,6.

Приклад 3. У десятковій системі застосовують числа 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, у вісімковій — 0, 1, 2, 3, 4, 5, 6, 7.

Приклад 4. У десятковому числі 725 вага другого розряду дорівнює $10^2 = 100$, а у числі 0,4563 вага розряду -3 дорівнює $10^{-3} = 0{,}001$.

Дані в комп'ютерних системах подаються символами 0 і 1. Це обумовлено тим, що у двійковій системі числення порівняно з іншими найпростіше виконуються арифметичні операції, і вона надійніше реалізується технічно.

Двійкова система (як і десяткова) є позиційною. Нагадаємо, що в таких системах кількісне значення цифри (кількісний еквівалент) залежить як від значення самої цифри, так і від її місця (позиції, розряду) у записі числа.

З урахуванням номера розряду й основи системи числення визначається кількісний еквівалент кожної цифри в числі (приклади 5, 6).

В обчислювальній техніці основною системою числення є двійкова система із символами 0 і 1. Застосовують також десяткову, вісімкову і шістнадцяткову системи числення.

Приклад 5. У десятковому числі 326,75 цифра 3 розміщена у другому розряді і має $3*10^2=300$ одиниць, цифра 2 розміщена у першому розряді і має $2*10^1=20$ одиниць, цифра 6 розміщена у нульовому розряді і має $6*10^0=6$ одиниць, цифра 7 розміщена в мінус першому розряді і має $7*10^{-1}=0{,}7$ одиниць, цифра 5 розміщена в мінус другому розряді і має $5*10^{-2}=0{,}05$ одиниць.

Десяткове число 326,75 можна записати у вигляді суми:

$$326{,}75=3*10^2+2*10^1+6*10^0+7*10^{-1}+5*10^{-2}$$

Приклад 6. Двійкове число 1011,01 можна записати у вигляді многочлена: $1*2^3+0*2^2+1*2^1+1*2^0+0*2^{-1}+1*2^{-2}=11{,}25$.

Еквівалентом двійкового числа 1011,01 є десяткове число 11,25.

Щоб визначити кількісне значення цифри в позиційній системі числення, необхідно помножити цю цифру на основу системи в степені того номера розряду, в якому розміщена ця цифра.

Крім перелічених систем числення, використовуються також інші системи, наприклад двійково-десяткова.

Двійково-десятковий код (англ. *binary-coded decimal*), BCD, 8421-BCD — форма запису раціональних чисел, коли кожний десятковий розряд числа записується у вигляді його чотирибітового двійкового коду.

У двійково-десятковій системі кожна десяткова цифра подається чотирма двійковими розрядами (приклад 7).

Приклад 7. Десяткове число 618,74 має такий запис у двійково-десятковій системі:

0110 0001 1000, 0111 0100 — двійково-десяткове значення

6 1 8 , 7 4

Крайні ліві нулі в цілій частині і крайні праві нулі у дробовій частині можна на папері не писати, тобто те саме десяткове число у двійково-десятковій системі можна записати так: 11000011000,011101.

Алан Кертіс Кей — американський інформатик, відомий своїми працями в галузі ООП, президент дослідного інституту В'юпоїнта, ад'юнкт-професор інформатики в Каліфорнійському університеті (США). Кей сказав, що найкращий спосіб спрогнозувати майбутнє — винайти його.

Чотири двійкові розряди називають тетрадою. Щоб знайти десятковий еквівалент двійково-десяткового числа, необхідно в шістнадцятковій цілій частині ліворуч від коми й у дробовій частині праворуч від коми відокремити тетради і знайти їхні десяткові значення. Неповні тетради умовно доповнюються нулями (приклад 8).

Щоб розрізняти, у якій системі задано на папері число, праворуч від нього в дужках інколи записують систему числення (приклад 9).

У шістнадцятковій системі значення її перших десяти символів збігаються з цифрами десяткової системи, а інші символи мають значення *A*, *B*, *C*, *D*, *E*, *F*. У табл. 5.1 наведено ці символи та відповідні їм двійкові та десяткові значення.

Приклад 8

100 1000 1001, 0101 001 — двійково-десяткове значення
4 8 9 , 5 2

Задане двійково-десяткове число дорівнює десятковому числу 489,52.

Приклад 9. $101\,001_{(2)}$; $10\,011\,001{,}011\,001_{(2\text{–}10)}$.

Перше число записане у двійковій системі числення, а друге — у двійково-десятковій системі.

Таблиця 5.1. Кодування шістнадцяткових символів

Шістнадцяткові символи	Двійкові значення	Десяткові значення
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Ми розглянули подання (кодування) десяткових та шістнадцяткових символів двійковими значеннями, у яких розряди мають значення (вагу) 8, 4, 2, 1. Такий код (8421) називають кодом прямого заміщення. Саме цей код є найрозповсюдженішим. Проте на практиці інколи використовують код із вагою розрядів 2421, а також код із залишком 3 (8421+3) та ін.

Запитання для перевірки знань

1. Який вигляд мають десяткові цифри у двійковій системі числення?
2. Як нумеруються розряди чисел?
3. Як записуються шістнадцяткові символи у двійковій системі числення?
4. Як записуються десяткові числа у вісімковій системі числення?
5. Чому двійкова система є основною в техніці?
6. Доведіть, чому не може бути двійково-десятковим число 10 1111 0111, 0111 1000.

Завдання для самостійного виконання

1. Запишіть десяткове число 93,64 у вигляді многочлена.
2. Запишіть двійкове число 110 111,011 у вигляді суми.
3. Запишіть десяткове число 170,25 у шістнадцятковій системі числення.
4. Шістнадцяткове число A0E,B запишіть у двійковій системі числення.
5. Чому число 1 011 101 101,001 001 01 не може бути двійково-десятковим?
6. Замініть розгорнутий запис двійкового числа $1*2^4+0*2^3+1*2^2+0*2^1+1*2^0+0*2^{-1}+1*2^{-2}$ на скорочений.
7. Запишіть двійково-десяткове число 1 011 000,001 001 01 у десятковій системі числення, а потім у вигляді суми.

5.3.2. Робота з великими числами

З якими великими числами вам доводилося зустрічатися у житті? Які, на вашу думку, проблеми можуть виникати в комп'ютері у процесі оброблення великих чисел?

У науці, техніці, побуті зустрічаються великі числа, тобто числа, що містять велику кількість розрядів, наприклад 20, 100 і більше (приклад 1). У математиці великі числа (а також досить малі) часто записують в експоненціальній формі, тобто з використанням мантиси і порядку (приклад 2).

Нормалізованими називають числа, у яких кома розміщується праворуч після першої значущої цифри.

У мові Python числа в експоненціальній формі записують так: замість числа 10 вказується велика буква E або мала буква e, а після букви — степінь. Знак множення не вказується, а замість коми ставиться крапка. Наприклад, 1.496E11 або 1.496e11.

У сучасних мовах програмування суттєвих проблем роботи з досить великими або досить малими числами майже не існує. Але слід враховувати, що математичні операції над великими або досить малими числами виконуються повільніше, ніж операції над невеликими числами. У мові Python фактично не існує обмежень на довжину цілих (тип int) і дійсних (тип float) чисел (приклад 3).

Приклад 1. До великих можна віднести такі назви чисел: мільярд $(1\,000\,000\,000 = 10^9)$, трильйон $(1\,000\,000\,000\,000 = 10^{12})$, квадрильйон $(1\,000\,000\,000\,000\,000 = 10^{15})$, квінтильйон $(1\,000\,000\,000\,000\,000\,000 = 10^{18})$ та ін.

Приклад 2. Десяткове число 149 600 000 000 може бути записано так: $1{,}496 \cdot 10^{11}$, або $0{,}001\,496 \cdot 10^{14}$, або $14{,}96 \cdot 10^{10}$. Перший запис означає, що число нормалізоване.

Приклад 3. Розглянемо подання двох 15-розрядних десяткових цілих чисел у мові Python і результат виконання над ними операції множення:

```
>>> a = 111222333444555
>>> b = 666777888999111
>>> a*b
74160592703715604789962790605
```

Подання двох дійсних чисел і результат виконання над ними операції множення пояснюється таким прикладом:

```
>>> a = 222333.444
>>> b = 555666.33
>>> a*b
123543208863.74051
```

Якщо перевірити точність обчислення цих чисел, то побачимо, що отриманий результат відрізняється від абсолютно точного на одну десятитисячну.

Якщо числа цілого або дійсного типу досить великі, то результат виконання операції над ними подається в експоненціальній формі:

```
>>> a = 111222333444555.666
>>> b = 555666777888999.11
>>> a*b
6.1802555654432115e+28
```

Проте навіть у тому випадку, якщо початкові числа подано в експоненціальній формі, результат виконання операції над ними може бути отриманий цілого або дійсного типу. Така ситуація виникає тоді, коли початкові числа мають невеликі значення.

Далі наведено такий варіант:

```
>>> a = 2.5e2
>>> b = 4.95e5
>>> a+b
495250.0
```

Для виконання обчислень із дійсними числами мова Python має значну кількість функцій у бібліотеці (модулі) math, які вивчаються в курсі 10 класу.

Отже, результат операції над цілими або дійсними числами подається в експоненціальній формі, якщо результат має досить велике або досить мале значення, і навпаки — у звичайній формі, якщо результат операції має не досить велике або не досить мале значення.

Нагадаємо, що для використання функцій необхідно на початку програмного коду підключити цю бібліотеку за допомогою команди import math.

Функція викликається командою math.<ім'я функції>(x), де x — це будь-яке число, змінна або вираз.

Функція повертає значення, яке можна вивести на екран, присвоїти іншій змінній або використати у виразі.

Приклад 4. Відомі максимальна і мінімальна відстані від Землі до Марса і швидкість польоту сучасного космічного корабля. Розробимо проект визначення кількості діб польоту до Марса. Варіант проекту зображено на рис. 5.4.

```
# За скільки діб можна долетіти до Марса?
max = 401000000                 # Максимальна відстань до Марса
min = 560000000                 # Мінімальна відстань до Марса
v = 58000                       # Швидкість космічного корабля
print ("Максимум за ", (max/v)/24, "діб")
print ("Мінімум за ", (min/v)/24, "діб")
input ()
```

Рис. 5.4. Програма визначення часу польоту до Марса

Результат виконання проекту:

```
Максимум за  288.0747126436782  діб
Мінімум за  40.229885057471265 діб
```

Запитання для перевірки знань

1. Які назви великих чисел вам відомі?
2. Як записуються числа мовою Python в експоненціальному вигляді?
3. У яких випадках результат виконання математичних операцій подається в експоненціальному вигляді?

Завдання для самостійного виконання

1. Число 4.375 помножте на 10 у степені 20. Результат виведіть на екран.
2. Розробіть програму перетворення градусів Фаренгейта у градуси Цельсія за допомогою формули C = 5/9*(F–32).
3. Перетворіть за допомогою функції **float()** рядок '234.25' у число. Доведіть, що отриманий результат є правильним.
4. Краща врожайність пшениці в Україні досягається при нормі висіву зерна 3–4 млн насінин, або 120–200 кг, на гектар. При цьому з кожного гектара збирають 3–3,5 тонни зерна. Розробіть програму визначення мінімального і максимального посіву зерна в насінинах і кілограмах на площі 100 га, а також мінімальний і максимальний врожай у насінинах і кілограмах.

5.3.3. Факторизація чисел

Які числа називають простими? На які прості множники можна розкласти числа 625 і 2100?

Факторизацією натурального числа називають його розкладання на добуток простих множників.

Наприклад, число 525 може бути розкладене на такі прості множники: $3*5*5*7=525$. Доведено, що кожне натуральне число має єдине розкладання на прості множники.

Тривіальним алгоритмом факторизації чисел є повний перебір можливих простих дільників, починаючи з числа 2. Його сутність така.

Крок 1	Задане натуральне число аналізується, чи не є воно простим. Якщо воно просте, факторизація завершується, а його множником є саме це число.
Крок 2	Якщо число складене, відшукується перший за величиною простий дільник, на який без залишку ділиться число (спочатку число 3, потім 5, далі 7, 11 і так далі).
Крок 3	Далі перевіряється простота отриманої частки. Якщо частка не є простим числом, факторизація продовжується у порядку, описаному вище, інакше — вона завершується.

Пояснимо сутність факторизації методом повного перебору на прикладі 1.

Приклад 1. Розглянемо сутність методу повного перебору на прикладі числа 84.

1. Визначаємо, що число 84 непросте, тому ділимо це число на 2. Отримаємо частку 42 і перший дільник 2.
2. Визначаємо, що число 42 непросте, тому ділимо його на 2. Отримаємо частку 21 і другий дільник 2.
3. Визначаємо, що число 21 непросте, тому робимо спробу поділити його на найменший дільник 2. Але число 21 не ділиться на 2. Тому переходимо до наступного простого числа.
4. Ділимо число 21 на 3. Отримаємо частку 7 і третій дільник 3.
5. Визначаємо, що число 7 просте. Тому четвертим простим дільником є 7. На цьому процес факторизації завершений.

Отже, число 84 можна розкласти на такі прості множники: $2*2*3*7=84$.

З описаного алгоритму можна зробити висновок, що для факторизації чисел натурального ряду методом повного перебору треба:

- уміти визначати, чи є частка на кожному кроці факторизації простим числом;
- уміти визначати черговий за величиною простий дільник.

RSA (абревіатура від прізвищ Rivest (Рівест), Shamir (Шамір) та Adleman (Адлеман)) — алгоритм криптографічної системи з відкритим ключем. Його надійність заснована на складності задач факторизації натуральних чисел. **RSA** став першим алгоритмом такого типу, придатним і для шифрування, і для цифрового підпису. Алгоритм **RSA** запропонували вчені Массачусетського технологічного інституту в 1977 р. Він має практичне значення у галузі шифрування і дозволяє факторизувати 100-розрядні числа.

На практиці прості числа деякого діапазону інколи зберігаються в таблиці (масиві, списку).

Сутність алгоритму розкладання цілого числа на прості множники з використанням масиву простих чисел шляхом повного перебору така:

Крок 1	Задане ціле число порівнюється зі значенням першого елемента масиву. Якщо вони рівні, то це означає, що саме число є простим. Інакше — робиться спроба поділити число на значення цього елемента масиву.
Крок 2	Якщо залишок дорівнює нулю, то множником є цей дільник, а частка ділиться на значення цього самого елемента масиву. Цей процес продовжується доти, поки залишок від ділення чергової частки на значення елемента масиву не буде дорівнювати нулю.
Крок 3	Якщо залишок не дорівнює нулю, вибирається черговий елемент масиву і далі продовжуються аналогічні процеси доти, поки остання частка не буде дорівнювати значенню елемента масиву.

Алгоритм факторизації натурального числа і його реалізацію на прикладі факторизації числа 90 наведено в табл. 5.2.

Таблиця 5.2. Послідовність виконання алгоритму факторизації

№	Алгоритм	Цикл					
		1-й	2-й	3-й	4-й	5-й	6-й
1	Створити mas1 = [2, 3, 5, 7]	mas1 = [2, 3, 5, 7]					
2	k1 — довжина масиву mas1	k1 := 4					
3	Створити порожній масив mas2	mas2[]					
4	Увести n	n := 90					
5	l := 0 (лічильник циклів)	l := 0					
6	Якщо n = mas1[i], то п. 7, інакше — п. 9	90 = 2, ні, п. 9		45 = 3, ні, п. 9			5 = 5, так, п. 7
7	Додати елемент mas1[i] до масиву mas2						mas2 [3] := 5
8	Перервати цикл (до п. 15)						до п. 15
9	Якщо n%mas1[i] = 0, то п. 10, інакше — п. 13	90%2 = 0, так, п. 10	45%2 = 0, ні, до п. 13	45%3 = 0, так, до п. 10	15%3 = 0, так, до п. 10	5%3 = 0, ні, до п. 13	
10	Додати mas1[i] до масиву mas2	mas2[0] := 2		mas2[1] := 3	mas2[2] := 3		
11	n := n//mas1[i]	n := 45		n := 15	n := 5		
12	Перейти до п. 9	до п. 9		до п. 9	до п. 9		
13	i := i+1		i := 1			i := 2	
14	Якщо i <= k1, то п. 6, інакше — п. 15		i <= 4, так, до п. 6			i <= 4, так до п. 6	
15	Виведення масиву mas2						Виведення 2 3 3 5

Програму реалізації алгоритму зображено на рис. 5.5.

```
# mas1 – це масив простих чисел
mas1 = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97]
mas2 = []                                   # Масив збереження простих множників
n = int(input ('Уведіть ціле число: '))     # Введення числа для факторизації
k1 = len(mas1)                              # Довжина масиву простих чисел
for i in range (k1):                        # Циклічне визначення простих множників
    if n == mas1[i]:                        # Якщо n просте число
        mas2.append (mas1[i])               # Збереження у масиві простого множника
        break                               # Переривання циклу
    else:                                   # Якщо n непросте число
        while ((n%mas1[i])== 0):            # Доти, доки частка ділиться на просте число
            mas2.append (mas1[i])           # Збереження чергового простого множника
            n = n//mas1[i]                  # Визначення чергової частки
k2 = len(mas2)                              # Довжина масиву простих множників
print ("Прості множники:")                  # Виведення повідомлення
for i in range(k2):                         # Циклічне виведення простих множників
    print (mas2[i], end = " ")              # Виведення результату
```

Рис. 5.5. Програма факторизації з використанням масиву простих чисел

Результат виконання програми:

```
Уведіть ціле число: 3267
Прості множники:
3 3 3 11 11
```

На рис. 5.6 наведено більш удосконалену програму факторизації натуральних чисел методом повного перебору, але без використання масиву простих чисел. У програмі використано ключове слово yield, яке виконує фактично ту ж функцію, що й return (повертає результат обчислення). У цьому випадку значення d передається у чергову позицію списку.

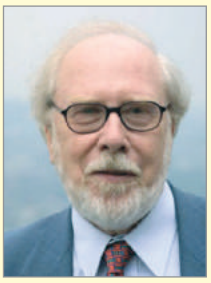
Наведений алгоритм є досить простим, але мало придатним для великих чисел, тому що він потребує збереження досить великих масивів простих чисел.

```
def func1 (x, d = 2):                       # Функція з двома параметрами
    while x>1:                              # Доти, доки x більше одиниці
# Функція divmode повертає частку q і залишок r від ділення x на d
        g, r = divmod (x, d)
        if r:                               # Якщо значення дорівнює True
            d += 1                          # Збільшити d на одиницю
        else:                               # Якщо значення дорівнює False
            yield d                         # Повернення значення d
            x = g                           # Параметр x набуває значення k1
n = int(input ("Уведіть ціле число "))      # Уведення числа n
# Звернення до функції func1, форматування і виведення результату
print ('{}={}'.format (n, ' * '.join(map(str, func1(n)))))
```

Рис. 5.6. Програма факторизації натуральних чисел із ключовим словом **yield**

Далі наведено один із варіантів виконання програми:

```
Уведіть ціле число: 2310
2310 = 2 * 3 * 5 * 7 * 11
```



Швейцарець Ніклаус Вірт — жива легенда в світі програмування. Це блискучий інженер і глибокий дослідник, який у 1984 році був удостоєний премії Тюрінга.

Зробимо пояснення щодо функції виведення. У програмі реалізовано форматоване виведення результату. У процесі форматування створюється рядок шляхом підстановки в нього даних, які отримуються у процесі виконання програми. Підстановка виконана за допомогою методу format. Якщо за допомогою цього методу потрібно підставити тільки один аргумент, то значенням є сам аргумент (приклад 1).

Приклад 1.

```
>>> 'Операційна {}'.format ('система')
'Операційна система'
```

Якщо для підстановки потрібні декілька аргументів, то значеннями є всі аргументи з рядками підстановки (звичайними або іменованими) (приклад 2).

Приклад 2.

```
>>> '{} {} {}'.format ('Операційна', 'система', 'Windows')
'Операційна система Windows'
>>> '{0} {1} {2}'.format ('Операційна', 'система', 'Windows')
'Операційна система Windows'
>>> '{2} {1} {0}'.format ('Операційна', 'система', 'Windows')
'Windows система Операційна'
```

Приклад 3.

```
>>> a = ['pop','rek']
>>> print (' '.join(a))
pop rek
>>> print ('**'.join(a))
pop**rek
>>> b = [23, 456]
```

У функції виведення використаний метод join, який дозволяє вивести список за допомогою однорядкової команди. Тут один параметр — список рядків. У результаті отримується рядок з'єднанням елементів списку (які передані як параметри) в один рядок. При цьому між елементами списку вставляється розмежувач, рівний тому рядку, в якому використовується метод. Розглянемо приклад 3.

Якщо список складається з чисел, то необхідно використати функцію map (приклад 4).

Приклад 4.

```
>>> b = [23, 456]
>>> print ('***'.join(map(str, b)))
23***456
```

Порядок виконання програми, яку подано на рис. 5.6 на прикладі факторизації числа 60 наведено в табл. 5.3.

Таблиця 5.3. Аналіз порядку виконання програми, зображеної на рис. 5.6

Інструкції	Пуск	Цикл							
		1-й	2-й	3-й	4-й	5-й	6-й	7-й	8-й
def func1 (x, d = 2)		x = 60, d = 2							
while x > 1		True	True	True	True	True	True	True	False
q, r = divmod(x, d)		q = 30, r = 0	q = 15, r = 0	q = 7, r = 1	q = 5, r = 0	q = 1, r = 2	q = 1, r = 1	q = 1, r = 0	
if r		False	False	True	False	True	True	False	
d += 1				d = 3		d = 4	d = 5		
Else									
yield d		2 у кортеж	2 у кортеж		3 у кортеж			5 у кортеж	
x = q		x = 30	x = 15		x = 5			x = 1	
n = int(input(«увести»))	n = 60								
print(' () = ()'.format(n'*'.join(map(str, func1(n)))))	Звернення до func1								60=2*2**3*5

Запитання для перевірки знань

1. Що називають факторизацією натурального числа?
2. Яке призначення має ключове слово **yield**?
3. Скільки різних варіантів простих множників може мати натуральне число?
4. Розкладіть на прості множники число 624.
5. Поясніть сутність факторизації цілих чисел методом повного перебору.
6. Сформулюйте алгоритм факторизації чисел з використанням масиву простих чисел.

Завдання для самостійного виконання

1. Проаналізуйте програму (рис. 5.5). Доведіть, для якого максимального числа вона гарантовано правильно реалізує факторизацію.
2. Виконайте кілька разів програму (рис. 5.5) для різних натуральних чисел. Для яких чисел програма надасть правильний результат? Обґрунтуйте, від чого це залежить.
3. Який необхідно мати мінімальний за довжиною масив простих чисел для факторизації числа 489? Обґрунтуйте свою відповідь.
4. Внесіть у програму (рис. 5.6) такі зміни, щоб прості множники виводилися не в рядок, а в один стовпець. Чи вдалося за рахунок цього спростити програму?
5. Розробіть таблицю виконання алгоритму факторизації числа 40 на основі методу повного перебору.
6. Розробіть таблицю виконання програми, зображеної на рис. 5.6, на прикладі факторизації числа 30.

6. Сортування і пошук даних

6.1. Алгоритми сортування даних

Що називають сортуванням даних?

Сортуванням називають процес упорядкування множини об'єктів за деякою ознакою, наприклад за збільшенням або зменшенням їх значень. Мета сортування — полегшити подальший пошук в упорядкованій множині.

Не існує універсального алгоритму сортування. Проте, маючи приблизні характеристики вхідних даних, можна дібрати метод, який працює оптимальним чином у даній ситуації (задачі).

Нині є десятки різних методів сортування, які відрізняються кількісними характеристиками показників. Для того щоб обґрунтовано зробити свій вибір, існують параметри, за якими проводиться оцінка алгоритмів.

Класифікація квадратичних алгоритмів сортування:
- сортування підрахунком;
- сортування вставленням (включенням);
- сортування вибором;
- сортування обміном.

Квадратичними називають алгоритми, у яких для повного сортування максимальна кількість операцій, які виконуються, дорівнює квадрату кількості його елементів. Тобто якщо масив містить n елементів, то для виконання алгоритму сортування потрібно виконати до n^2 операцій.

Розглянемо алгоритми сортування даних вибором та обміном.

6.1.1. Сортування вибором

Припустимо, що в довільному порядку розташовані гирі вагою 20, 5, 100, 500 і 200 грамів. Як розмістити їх у порядку збільшення ваги шляхом поступового вибору найважчої з тих, що залишилися?

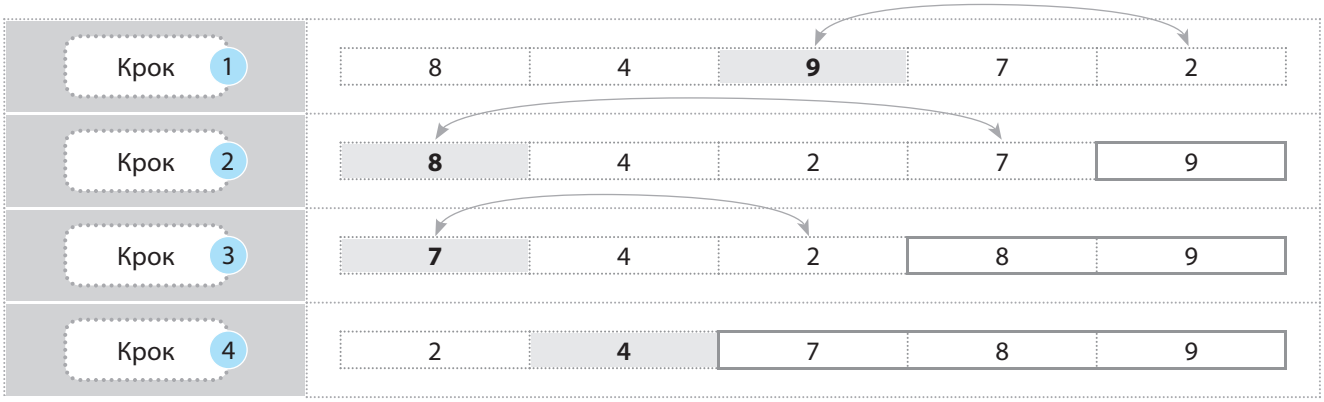
Нехай дано масив: mas[0], mas[1], ... , mas[n], який необхідно упорядкувати в порядку зростання значень його елементів.

Розглянемо сутність алгоритму сортування вибором.

Крок 1	Знаходять елемент із максимальним значенням і міняють його місцем з останнім елементом масиву.
Крок 2	Останній елемент із подальшого розгляду виключають, а для перших n – 1 елементів процедуру повторюють. Тобто аналізується масив mas[0], mas[1], ..., mas[n – 1], у якому також відшукують максимальний елемент. Цей елемент міняють місцями з елементом mas[n – 1].
Крок 3	Подібні дії виконують у масиві mas[0], mas[1], ..., mas[n – 2], потім у масиві mas[0], mas[1], ..., mas[n – 3] і т. д.

Процес упорядкування масиву чисел розглянемо на прикладі 1.

Приклад 1. Упорядкуємо масив чисел: 8, 4, 9, 7, 2.



Отже, на кожному кроці виконання алгоритму значення правої межі невпорядкованої частини масиву зменшується на одиницю (поточне максимальне значення елемента показано жирним шрифтом).

У програмі поточне значення правої межі будемо зберігати у змінній *p*.

Ознайомимося з алгоритмом сортування масиву обміном, який можна подати так.

Крок 1	Створення масиву mas.
Крок 2	Визначення довжини масиву n.
Крок 3	p = n – 1 # Початкове значення правої межі
Крок 4	Виконувати п. 5–11 доти, доки p > 0, інакше — п. 12.
Крок 5	m = 0 # Початковий номер максимального елемента (номер першого елемента масиву)
Крок 6	i = 1 # Номер другого елемента масиву
Крок 7	Якщо mas[i] > mas[m], то m = i, інакше — п. 8.
Крок 8	i = i + 1 # Перехід до чергового елемента
Крок 9	Якщо i <= p, то п. 7, інакше — п. 10.
Крок 10	Поміняти місцями максимальний елемент із поточним правим крайнім.
Крок 11	p = p – 1 # Зменшення значення правої межі
Крок 12	Виведення упорядкованого масиву mas.
Крок 13	Кінець.

Програму реалізації алгоритму зображено на рис. 6.1.

```
# Сортування масиву методом вибору
mas = [21, 40, 3, 33, 17, 9]      # Початковий масив
n = len(mas)                      # Довжина масиву
p = n-1                           # Права межа масиву
while p>0:                        # Доти, доки права межа більша від нуля
    m = 0                         # Початковий номер максимального елемента
    for i in range (p):           # Внутрішній цикл перегляду масиву
        if mas [i]>mas[m]:        # Поточний елемент більше за максимальний?
            m = i                 # Номер максимального елемента
# Обмін елементів
    z = mas[m]; mas[m] = mas[p]; mas[p] = z
    p = p-1                       # Зменшення значення правої межі
for i in range (n):               # Циклічне виведення елементів
print (mas[i], end =" ")          # упорядкованого масиву
```

Рис. 6.1. Програма упорядкування масиву методом вибору

У результаті виконання програми отримаємо:

```
3 9 17 21 33 40
```

Перевага описаного методу сортування полягає в його простоті. Проте він є найповільнішим (не враховується те, що в заданому масиві деякі елементи можуть бути впорядковані). Навіть якщо певний масив буде повністю впорядкованим, кількість операцій порівняння в алгоритмі не зменшиться.

6.1.2. Сортування обміном

У шеренгу стали 5 учнів. Порівняйте зріст перших двох. Якщо перший вищий за другого, поміняйте їх місцями, за потреби — другого і третього і т. д. Як розмістяться учні?

Розглянемо сутність алгоритму сортування масиву в порядку зростання значень його елементів методом обміну.

Крок 1	Масив mas[0], mas[1], ... , mas[n] переглядається зліва направо. Спочатку порівнюються елементи mas[0] і mas[1], потім — mas[1] і mas[2], mas[2] і mas[3], mas[3] і mas[4] і так далі аж до елементів mas[n – 1] і mas[n]. Щоразу, коли попередній елемент більший від наступного, елементи міняються місцями. Зрозуміло, що після повного завершення першого перегляду всього масиву на останній позиції перебуватиме елемент із максимальним значенням.
Крок 2	Після цього елемент mas[n] із подальшого розгляду виключається, переглядається масив mas[0], mas[1], …, mas[n – 1] і над його елементами виконуються аналогічні дії. У результаті на позицію n – 1 буде переміщено другий за величиною елемент.
Крок 3	Процедура повторюється для масиву mas[0], mas[1], …, mas[n – 2], потім — для масиву mas[0], mas[1], …, mas[n – 3] і т. д.

З описаного випливає, що ознакою упорядкованості масиву є те, що після завершення його перегляду жодної перестановки елементів не було.

У прикладі 2 наведено порядок переміщення максимального числа на крайню праву позицію у масиві [4, 2, 5, 7, 6, 1]. Жирним шрифтом виділено елементи, які порівнюються.

Приклад 2. Перемістимо на крайню праву позицію максимальний елемент у масиві [4, 2, 5, 7, 6, 1].

Крок 1	**4**	**2**	5	7	6	1
Крок 2	2	**4**	**5**	7	6	1
Крок 3	2	4	**5**	**7**	6	1
Крок 4	2	4	5	**7**	**6**	1
Крок 5	2	4	5	6	**7**	**1**
Крок 6	2	4	5	6	1	7

Отже, після першого перегляду масиву елемент із максимальним значенням буде розташований на останній позиції.

Розробимо алгоритм сортування методом обміну.

В алгоритмі використано такі позначення: p — індекс правої межі поточної ділянки масиву; y — ознака наявності перестановки: на початку кожного зовнішнього циклу y набуває значення True.

Якщо після завершення зовнішнього циклу вона має значення True, це означає, що перестановок під час останнього перегляду не було; якщо y = False, то відбулася принаймні одна перестановка; i — індекс поточного елемента масиву; z — змінна, призначена для тимчасового зберігання значення елемента масиву під час перестановки елементів.

Розглянемо алгоритм сортування масиву методом обміну, який може бути таким.

Крок 1	Створення масиву mas.	Крок 7	Якщо mas[i] > mas[i+1], то поміняти елементи місцями і y = False.
Крок 2	Визначення довжини масиву n.	Крок 8	i = i + 1.
Крок 3	p = n–1 (початкове значення правої межі).	Крок 9	Якщо i < p, то п. 7, інакше — п. 10.
Крок 4	Виконувати п. 5–10 доти, доки p > 0, інакше — п. 11.	Крок 10	p = p – 1.
Крок 5	y=True (ознака відсутності переміщення елементів).	Крок 11	Виведення упорядкованого масиву mas.
Крок 6	$i=1$.	Крок 12	Кінець.

Програму реалізації алгоритму зображено на рис. 6.2.

```
# Сортування масиву методом обміну
mas = [51, 16, 63, 7, 100, 20, 31]    # Початковий масив
n = len(mas)                          # Довжина масиву
p = n-1                               # Поточне значення правої межі
while p>0:                            # Доти, доки значення правої межі більше за 0
    y = True                          # Ознака відсутності переміщення елементів
    for i in range (p):               # Внутрішній цикл перегляду масиву
        if mas [i]>mas[i+1]:          # Попередній елемент більший за наступний?
# Переміщення максимального елемента в поточну праву позицію
            z = mas[i]; mas[i] = mas[i+1]; mas[i+1] = z
            y = False                 # Ознака наявності переміщення у внутрішньому циклі
    p = p-1                           # Зменшення значення правої межі масиву
for i in range (n):                   # Циклічне виведення значень
    print (mas[i], end =" ")          # елементів вісортованого масиву
```

Рис. 6.2. Програма упорядкування масиву методом обміну

Результат виконання програми:

```
7 16 20 31 51 63 100
```

Запитання для перевірки знань

1. Як здійснюється обмін значеннями між двома змінними?
2. Які існують найпростіші методи сортування масиву?
3. Поясніть сутність алгоритму сортування масиву методом вибору.
4. Поясніть сутність алгоритму сортування масиву методом обміну.
5. Наведіть приклад алгоритму сортування масиву методом вибору.
6. Наведіть приклад алгоритму сортування масиву методом обміну.

Завдання для самостійного виконання

1. Проаналізуйте порядок виконання алгоритму сортування вибором на прикладі масиву 31, 20, 30, 35.
2. Визначте кількість операцій порівняння в алгоритмі сортування методом обміну в процесі впорядкування масиву 100, 20, 41, 30, 35.
3. Дано масив чисел: 34, 12, 8, 5, 20, 17. Виконайте сортування заданого масиву за допомогою методу обміну в порядку зменшення чисел.
4. Дано масив: 'процесор', 'команда', 'флешка', 'брелок', 'клавіатура'. Виконайте сортування його елементів в алфавітному порядку за допомогою методу вибору.
5. Розробіть програму створення масиву з 10 випадкових чисел у діапазоні від 1 до 8 та його сортування за допомогою методу обміну.
6. Розробіть програму створення масиву з 12 випадкових чисел у діапазоні від 4 до 11 і сортування за допомогою методу вибору.

6.1.3. Сортування вставленням

Як у впорядкований одновимірний масив чисел можна вставити нове число, не порушуючи впорядкування масиву?

Метод сортування вставленням використовується для впорядкованих масивів. Розглянемо сутність алгоритму сортування вставленням детальніше.

Крок 1

Нехай дано впорядкований у порядку зростання значень елементів масив чисел:

3	5	6	8	9	12

Крок 2

У масив потрібно вставити новий елемент зі значенням 7. Відшукуємо позицію, у яку слід уставити число 7, не порушуючи впорядкованості елементів масиву. Зрозуміло, що це позиція між числами 6 і 8. Зсуваємо усі елементи, починаючи з елемента 8, на одну позицію праворуч:

3	5	6		8	9	12

Крок 3

Уставляємо у вільну позицію число 7:

3	5	6	7	8	9	12

На рис. 6.3 наведено програму, яка реалізує вставлення нового елемента в початковий масив.

```
mas = [3, 5, 6, 8, 9, 12, 99]                # Масив упорядкованих чисел
n = len(mas)-1                               # Довжина масиву
c = int(input ('Увести новий елемент:'))     # Число для вставлення у масив
i = 0                                        # Початковий номер елемента масиву
while (c >= mas[i]):                  # Доти, доки число с більше значень елементів масиву
    i = i+1                           # Збільшення поточного номера елемента масиву
k = n-i                               # Кількість елементів масиву для зсуву
p = n-1                               # Номер останнього елемента масиву для зсуву
for j in range (k):                   # Цикл зсуву елементів масиву
    mas[p+1] = mas[p]                 # Зсув елементів масиву
    p = p-1                           # Зміна поточного номера елемента масиву для зсуву
mas[i] = c                            # Вставлення у масив уведеного числа
for i in range (n+1):                 # Циклічне виведення елементів нового масиву
    print (mas[i], end =' ')
```

Рис. 6.3. Програма сортування масиву методом вставлення

Зверніть увагу, що в програмі використано початковий масив [3, 5, 6, 8, 9, 12], хоча оголошено масив [3, 5, 6, 8, 9, 12, 99]. Довжина оголошеного масиву збільшена на одиницю, й останній елемент містить число 99. Це обумовлено тим, що після вставлення масив за довжиною буде більшим на одиницю, чим забезпечується правильна індексація елементів масиву.

Результат виконання програми:

```
Увести новий елемент: 7
3 5 6 7 8 9 12
```

6.1.4. Сортування злиттям

Припустимо, що є одновимірний масив цілих чисел завдовжки 20. Чи можна, на вашу думку, виконати його сортування, якщо спочатку виконати сортування перших 10 чисел, потім останніх 10 чисел? Як це можна зробити?

Сортування злиттям належить до зовнішніх методів сортування. Він був створений Джоном фон Нейманом у 1945 році та вважається одним із найпростіших алгоритмів серед швидких алгоритмів сортування.

Розглянемо сутність алгоритму сортування масиву злиттям.

Крок 1	Заданий масив ділиться приблизно на дві рівні частини. Виконується сортування кожної частини окремо, наприклад у порядку збільшення значень її елементів.
Крок 2	Потім вибираються елементи першої і другої частин масиву, розташовані на лівих крайніх позиціях. Менший із цих елементів записується у першу позицію буфера. А частина масиву, з якої вибрано менший елемент, зсувається на одну позицію ліворуч.
Крок 3	На наступному кроці теж порівнюються елементи обох частин, розташованих на лівій крайній позиції. Також вибирається менший із них і заноситься у чергову позицію буфера. Цей процес продовжується доти, поки одна із частин масиву стане порожньою. Після цього в буфер без змін у порядку розташування переносяться всі значення елементів тієї частини, у якій залишилися елементи.

Розглянемо алгоритм сортування масиву злиттям на прикладі.

Приклад. Нехай дано масив цілих чисел: 12, 7, 15, 4, 11, 1, 9.

1. Ділимо масив на дві частини, наприклад, так: перша частина 12, 7, 15, 4; друга частина 11, 1, 9.
2. Упорядковуємо частини в порядку збільшення значень елементів: перша частина 4, 7, 12, 15; друга — 1, 9, 11.
3. Вибираємо і порівнюємо елементи, розташовані на крайніх лівих позиціях (числа 4 і 1).

   Менше значення записуємо у ліву крайню позицію буфера.

4	7	12	15
1	9	11	

1						

Відомий фахівець у галузі інформатики Д. Кнут писав, що, навіть якби сортування було марним, знайшлася б маса причин зайнятися ним. Винахідливі методи сортування свідчать про те, що воно саме по собі цікаве як об'єкт дослідження.

4. Зсуваємо другу частину масиву на одну позицію ліворуч. Порівнюємо числа 4 і 9. Число 4 записуємо у чергову позицію буфера.

4	7	12	15
9	11		

1	4					

5. Зсуваємо першу частину на одну позицію ліворуч. Порівнюємо числа 7 і 9. Число 7 записуємо у чергову позицію буфера.

7	12	15	
9	11		

1	4	7				

6. Зсуваємо першу частину на одну позицію ліворуч. Порівнюємо числа 12 і 9. Число 9 записуємо у буфер.

12	15		
9	11		

1	4	7	9			

7. Зсуваємо другу частину на одну позицію ліворуч. Порівнюємо числа 12 і 11. Число 11 записуємо у буфер.

12	15		
11			

1	4	7	9	11		

8. Зсуваємо другу частину на одну позицію ліворуч. Оскільки друга частина стала порожньою, записуємо всі елементи, що залишилися у першій частині, у буфер без змін.

12	15		

1	4	7	9	11	12	15

На цьому процес сортування завершено.

Відзначимо, що алгоритм сортування злиттям для великих масивів є одним із найбільш швидкісних. Але він потребує додаткової пам'яті для збереження буферу, який за обсягом рівний обсягу початкового масиву.

Сортування підрахунком — це алгоритм, у якому використовується діапазон чисел сортованого масиву для підрахунку співпадаючих елементів. Метод є доцільним лише тоді, коли сортовані числа мають діапазон можливих значень, який досить малий у порівнянні з множиною, яка сортується.

В основі **сортування вставленням** лежить пошук для чергового елемента масиву відповідного місця у відсортованій частині із наступним включенням його в знайдену позицію.

В основу **сортування вибором** покладено вибір відповідного елемента для певної позиції в масиві.

Сортування обміном базується на повторенні етапів порівняння сусідніх ключів при русі вздовж масиву. Якщо наступний елемент виявиться меншим від попереднього, то відбувається обмін (звідси і назва методу).

Програму сортування масиву злиттям зображено на рис. 6.4.

```
mas = [20, 7, 17, 50, 4, 11, 44, 2, 9, 44]          # Початковий масив
n = len(mas)                                        # Довжина початкового масиву
k1 = n//2                                           # Довжина лівої частини масиву
k2 = n-k1                                           # Довжина правої частини масиву
mas1 = []                                           # Масив для збереження лівої частини
mas2 = []                                           # Масив для збереження правої частини
for i in range (k1):                                # Циклічне формування і збереження
    mas1.append (mas[i])                            # лівої частини масиву
mas1.sort()                                         # Сортування лівої частини масиву
for i in range (k1, n, 1):                          # Циклічне формування і збереження
    mas2.append (mas[i])                            # правої частини масиву
mas2.sort()                                         # Сортування правої частини масиву
mas.clear()                                         # Очищення початкового масиву
while k1>0 and k2>0:                                # Доти, доки є елементи в обох частинах
    if mas1[0] <= mas2[0]:                          # Якщо елемент першої частини менший від елемента другої
        mas.append (mas1[0])                        # Злиття частин масиву
        k1 = k1-1                                   # Зменшення кількості елементів у першій частині
        i = 0                                       # Підготовка до виконання циклу
        while i<k1:                                 # Доти, доки є елементи у першій частині
            mas1[i] = mas1[i+1]                     # Зсув елементів першої частини
            i = i+1                                 # Підготовка до зсуву чергового елемента
    else:                                           # Елемент першої частини більший за елемент другої
        mas.append (mas2[0])                        # Злиття частин масиву
        k2 = k2-1                                   # Зменшення кількості елементів у другій частині
        i = 0                                       # Підготовка до виконання циклу
        while i<k2:                                 # Доти, доки є елементи в другій частині
            mas2[i] = mas2[i+1]                     # Зсув елементів другої частини
            i = i+1                                 # Підготовка до зсуву чергового елемента
if k1 == 0:                                         # Якщо у першій частині не залишилося елементів
    mas.extend (mas2)                               # Злиття частин масиву
else:                                               # Якщо у другій частини не залишилося елементів
    mas.extend (mas1)                               # Злиття частин масиву
for i in range (n):                                 # Циклічне виведення
    print (mas[i], end =' ')                        # упорядкованого масиву
```

Рис. 6.4. Програма сортування масиву злиттям

Результат виконання програми:

```
2  4  7  9  11  17  20  44  44  50
```

6.1.5. Сортування підрахунком

Нехай є масив цілих чисел завдовжки 18, елементи якого мають значення 0, 1, 2. Які особливості слід врахувати в процесі впорядкування такого масиву?

Метод сортування підрахунком застосовується тільки для масивів цілих чисел. Із точки зору швидкодії й необхідності використання додаткової пам'яті цей метод є ефективним для масивів, у яких зберігаються числа невеликого діапазону, особливо для чисел у діапазоні від 0 до 9.

Розглянемо сутність алгоритму сортування підрахунком на прикладі.

Приклад

Дано початковий масив чисел, який потрібно відсортувати в порядку зростання значень його елементів:

2	0	0	1	5	3	1	2	0	3	0	1	2	1	0

1. Створимо додатковий порожній масив, довжина якого на одиницю більша від максимального значення елемента в початковому масиві (у наведеному масиві максимальним є число 5):

0	0	0	0	0	0

2. Значення елементів початкового масиву аналізуються зліва направо. Після аналізу кожного елемента відповідний елемент додаткового масиву збільшується на одиницю. Значення числа елемента початкового масиву визначає номер елемента додаткового масиву, який повинен збільшуватися на одиницю. Наприклад, якщо значення елемента початкового масиву дорівнює 3, то третій елемент додаткового масиву повинен збільшитися на одиницю (нумерація елементів масиву починається з нуля). Після завершення аналізу всіх елементів початкового масиву додатковий масив набуде такого змісту:

5	4	3	2	0	1

3. Початковий масив заповнюється новими значеннями, починаючи з нульової позиції. Оскільки перший елемент додаткового масиву має значення 5, то підряд записуються п'ять нулів, далі чотири одиниці і т. д. Якщо значення елемента додаткового масиву дорівнює 0, то відповідне число в масив не записується.

Один із найпростіших варіантів реалізації алгоритму сортування підрахунком зображено на рис. 6.5. Програма виконує сортування масиву цілих невід'ємних чисел (від 0 до 9) з максимальним значенням елемента не більше 9. Для сортування чисел із більшим діапазоном необхідно збільшити на потрібну величину розмір додаткового масиву, тобто внести зміни до третьої зверху інструкції: mas1 = [0, 0, …, 0].

```
# Сортування масиву підрахунком
mas = [2, 0, 0, 1, 3, 3, 1, 2, 0, 3, 0, 1, 2, 1, 0]      # Початковий масив
n = len(mas)                          # Довжина початкового масиву
mas1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]                    # Створення додаткового масиву
for i in range (n):                   # Цикл формування додаткового масиву
    j = mas[i]                        # Номер позиції додаткового масиву
    mas1[j] = mas1[j]+1               # Збільшення значення елементів додаткового масиву
mas.clear ()                          # Очищення початкового масиву
n1 = len(mas1)                        # Довжина додаткового масиву
p = -1                                # Початкове значення цифри для запису в масив
for i in range (n1):                  # Зовнішній цикл створення упорядкованого масиву
    k = mas1[i]                       # Кількість елементів і-тої позиції додатк. масиву
    p = p+1                           # Поточне значення цифри для запису в масив
    while k>0:                        # Внутрішній цикл створення упорядкованого масиву
        mas1.append (p)               # Додавання у масив чергового значення
        k = k-1                       # Зменшення поточного значення кількості цифр
for i in range (n):                   # Циклічне виведення значень
    print (mas[i], end =" ")          # елементів упорядкованого масиву
```

Рис. 6.5. Програма сортування масиву підрахунком

У результаті виконання програми отримаємо:

0	0	0	0	0	1	1	1	1	2	2	2	3	3	5

Один із засновників кібернетики Н. Вірт, автор відомої книги «Алгоритми і структури даних», писав: «Складається враження, що можна побудувати цілий курс програмування, вибираючи приклади тільки із завдань сортування».

Зверніть увагу на те, що в початковому масиві відсутнє число 4. Але в додатковому масиві для нього відведено позицію, значення елемента якої дорівнює нулю. У програмі така ситуація врахована. Позиції додаткового масиву з нульовим значенням ігноруються, тому і число 4 у відсортований масив не виводиться.

Методи сортування найчастіше аналізуються за такими характеристиками:

- кількість порівнянь в ітерації;
- загальна кількість порівнянь;
- кількість ітерацій (переглядів).

Конкретні значення цих характеристик дозволяють вибрати оптимальний метод сортування для певного завдання.

Деякі дослідники надають перевагу сортуванню методом обміну, оскільки він є простим і для роботи з елементами масиву використовує тільки порівняння.

Запитання для перевірки знань

1. Для яких масивів можна виконувати сортування вставками?
2. Для яких масивів доцільно використовувати сортування підрахунком?
3. Як можна поділити масив на дві приблизно рівні частини?
4. Наведіть приклад сортування масиву злиттям.
5. Поясніть сутність алгоритму сортування масиву вставленням.
6. Поясніть сутність алгоритму сортування масиву злиттям.
7. Поясніть сутність алгоритму сортування масиву підрахунком.
8. За якими характеристиками зазвичай аналізуються методи сортування?

Завдання для самостійного виконання

1. Проаналізуйте порядок виконання програми сортування масиву методом вставлення (рис. 6.3) на прикладі вставлення числа 13 у масив [7, 10, 15, 21, 29, 39].
2. Проаналізуйте порядок виконання програми сортування масиву методом злиття (рис. 6.4) на прикладі масиву [11, 75, 82, 40, 5, 15].
3. Проаналізуйте порядок виконання програми сортування масиву методом підрахунку (рис. 6.5) на прикладі масиву [3, 0, 0, 1, 0, 1, 3, 3, 3].
4. Обґрунтуйте, який алгоритм сортування масиву [6, 2, 5, 8, 20, 27, 30, 41] у порядку збільшення значень його елементів доцільно вибрати, щоб забезпечити мінімальний час його виконання.
5. На основі аналізу програми (рис. 6.5) подайте у словесно-формульній формі алгоритм сортування масиву підрахунком.
6. На основі здійснення аналізу програми (рис. 6.4) подайте у словесно-формульній формі алгоритм сортування масиву злиттям.

6.2. Алгоритми пошуку даних

Завдання пошуку даних можна сформулювати так: знайти у множині даних один або декілька елементів, які відповідають заданим властивостям. Пошук даних виконується в різних структурах даних, наприклад у словниках, списках, базах даних, масивах та інших.

Вибір того чи іншого алгоритму пошуку безпосередньо залежить від структури даних, для якої він реалізований. Далі описано алгоритми і приклади програм пошуку даних у масивах.

Пошук потрібної інформації може здійснюватись в упорядкованому наборі та в неупорядкованому. Існують алгоритми пошуку даних, які класифікують на основі механізму пошуку.

Класифікація алгоритмів пошуку даних:
- послідовний (лінійний) пошук;
- бінарний (двійковий) пошук;
- пошук із поверненням;
- тернарний пошук.

6.2.1. Послідовний пошук

Припустимо, що в коробці лежать кулі з номерами від 1 до 21. Який алгоритм пошуку кулі з номером 13 можна запропонувати?

Послідовний пошук базується на прямому переборі елементів у невпорядкованому масиві, наприклад зліва направо, і порівнянні кожного з них із заданим значенням.

Нехай дано масив a[0], a[1], ..., a[n] і потрібно визначити, чи є у цьому масиві елемент, значення якого збігається зі значенням *c*. Зрозуміло, що в масиві може бути декілька значень, які збігаються зі значенням *c*. Але будемо вважати, що потрібно визначити лише сам факт наявності елемента *c* в масиві, тобто пошук припиняється одразу після знаходження першого такого елемента.

Сутність алгоритму послідовного пошуку така.

Крок 1	Спочатку значення **c** порівнюється зі значенням a[0]. Якщо вони збігаються, робиться висновок, що елемент знайдено на нульовій позиції масиву і на цьому пошук завершується.
Крок 2	Інакше — значення **c** порівнюється зі значенням a[1] і робиться аналогічний висновок.
Крок 3	Далі аналогічний процес може повторюватися над елементом a[2] і так далі до a[n].

Розглянемо сутність алгоритму послідовного пошуку на прикладах 1, 2.

Приклад 1. У міжнародному марафоні беруть участь представники 8 країн. Кількість учасників не обмежена. Учасники від кожної країни мають власний номер і номер країни. Потрібно розробити алгоритм та програму визначення, чи є серед 12 учасників, які фінішували першими, представник команди за номером 5.

Цю задачу можна формалізувати таким чином.

Результати масових змагань учасників можна вважати випадковими. Тому сформуємо одновимірний масив цілими випадковими числами в діапазоні від 1 до 8 завдовжки 12 і визначимо першу позицію елемента, на якій розташоване число 5.

Розглянемо один із варіантів алгоритму, його подано в словесно-формульній формі.

Крок 1	Сформувати масив mas із 12 випадковими числами у діапазоні від 1 до 8.	Крок 6	Перехід на п. 9.
Крок 2	*p* = 'Учасника цієї країни немає'	Крок 7	*i* = *i* + 1.
Крок 3	*i* = 0.	Крок 8	Якщо *i* <= 12, то п. 4, інакше — п. 9.
Крок 4	Якщо mas[*i*] = 5, то п. 5, інакше — п. 7.	Крок 9	Виведення p.
Крок 5	p = 'Учасник цієї країни фінішував', *i* + 1.	Крок 10	Кінець.

Програму реалізації алгоритму зображено на рис. 6.6.

```
# Послідовний пошук в одновимірному масиві
import randome                          # Імпортування модуля randome
a = []                                  # Порожній масив
for i in range (12):                    # Цикл формування масиву
# Генерування випадкових чисел від 1 до 8
    b = randome.randint (1, 8)
    a.append (b)                        # Додавання випадкового числа до масиву
    print (a[i], end =" ")              # Виведення елементів масиву
p = "Учасника цієї країни немає"
print ()                                # Друк із нового рядка
for i in range (12):                    # Циклічний перегляд масиву
    if a[i] == 5:                       # Поточний елемент дорівнює 5?
        p = ("Учасник цієї країни фінішував", i+1)
        break                           # Переривання циклу
print (p)                               # Виведення результату
```

Рис. 6.6. Послідовний пошук елемента в одновимірному масиві

Один із можливих результатів виконання програми:

```
8 5 1 6 5 2 1 3 6 3 1 8
 Учасник цієї країни фінішував 2
```

Інколи необхідно визначити не тільки наявність заданого елемента в масиві, а й знайти усі ці елементи, наприклад для обчислення їх суми, кількості таких елементів тощо. У такому разі аналізуються всі елементи масиву.

Приклад 2. Розробимо програму для завдання, що сформульовано у прикладі 1, але визначимо кількість учасників країни, зареєстрованої під номером 5, які фінішували в числі перших 12.

Алгоритм розв'язування цього завдання несуттєво відрізняється від попереднього, а програму його реалізації зображено на рис. 6.7.

```
# Обчислення кількості заданих елементів у масиві
import randome                          # Імпорт модуля randome
mas = []                                # Порожній масив
for i in range (12):                    # Циклічне формування масиву
# Генерування випадкових чисел у діапазоні від 1 до 8
    b = randome.randint (1, 8)
    mas.append (b)                      # Додавання випадкового числа до масиву
k = 0                                   # Початкова кількість учасників за № 5
for i in range (12):                    # Циклічний перегляд елементів масиву
    print (mas[i], end = " ")           # Виведення чергового елемента масиву
    if mas[i] == 5:                     # Чи дорівнює поточний елемент 5?
        k = k+1                         # Підрахунок кількості чисел, рівних 5
print ()
print ("Від країни за номером 5 фінішувало учасників", k)
```

Рис. 6.7. Обчислення кількості елементів в одновимірному масиві

Один із можливих результатів виконання програми може бути таким:

```
3 4 3 3 5 2 5 8 1 3 4 1
Від країни за номером 5 фінішувало учасників 2
```

6.2.2. Бінарний пошук

Які, на вашу думку, особливості пошуку елементів в упорядкованому масиві слід враховувати з метою прискорення процесу пошуку?

Бінарний пошук можна застосовувати лише до впорядкованих масивів.

Нехай дано масив mas[1], ..., mas[*n*], елементи якого впорядковані за зростанням їхніх значень, і ключове значення *c*, яке потрібно знайти в масиві. Пошук будемо здійснювати методом половинного ділення.

Сутність алгоритму бінарного пошуку така.

Крок 1	Позначимо поточне значення лівої межі масиву змінною l, а значення правої межі — змінною p (початкові значення змінних: i = 1 і p = n).
Крок 2	Спочатку в масиві обирають елемент, розташований усередині масиву, — mas[i]. Значення індексу середнього елемента можна визначити за формулою i = [(l + p) / 2] (квадратними дужками позначено цілу частину числа).
Крок 3	Значення середнього елемента порівнюють із ключовим значенням c. Якщо c = mas[i], елемент знайдено. Якщо c < mas[i], то далі для пошуку вибирають частину масиву, розташовану ліворуч від mas[i], у протилежному випадку — частину масиву, розташовану праворуч від mas[i]. Для вибраної частини процес повторюють.

Розглянемо алгоритм бінарного пошуку на прикладі 3.

Приклад 3. Нехай дано масив комп'ютерних термінів: алгоритмізація, біт, змінна, масив, миша, файл. Розробимо програму визначення, чи є в цьому масиві термін, який після запуску програми вводиться з клавіатури. Далі наведено алгоритм розв'язування.

Крок 1	Сформувати масив mas комп'ютерних термінів.	Крок 7	Виконувати пункти 8–10 доти, доки *l* <= *p* and not per, інакше — п. 11.
Крок 2	Визначити довжину n масиву.	Крок 8	*i* = (*l*+*p*)/2 # Номер середнього елемента
Крок 3	Увести у змінну *c* — термін для пошуку.	Крок 9	Якщо mas[*i*] = *c*, то per = True (ознака наявності терміна), інакше — п. 10.
Крок 4	per = False # Ознака відсутності терміна	Крок 10	Якщо *c* > mas[*i*], то *l* = *i*+1 (зміна лівої межі), інакше — *p* = *i*–1.
Крок 5	*l* = 0 # Початкове значення лівої межі.	Крок 11	Якщо per = True, то виведення — "Термін на позиції", *i*; інакше — виведення "Термін відсутній".
Крок 6	*p* = *n*–1 # Початкове значення правої межі		

Програму реалізації алгоритму зображено на рис. 6.8.

```
# Бінарний пошук в одновимірному масиві
import math                                  # Імпортування модуля math
mas = ['алгоритмізація', 'біт', 'змінна',    # Порожній масив
       'код', 'масив', 'миша', 'файл']
n = len(mas)                                 # Довжина масиву
c = str(input ("Увести термін:"))            # Уведення терміна для пошуку
per = False                                  # Ознака відсутності терміна в масиві
l = 0                                        # Початкове значення лівої межі
p = n-1                                      # Початкове значення правої межі
while ((l <= p) and (not per)):              # Доти, доки l <= p and not per
    i = int (math.floor (l+p)/2)             # Номер середнього елемента
    if mas[i] == c:                          # Термін масиву співпадає із введеним?
        per = True                           # Ознака наявності терміна
        break                                # Переривання циклу
    else:                                    # Термін не співпадає із введеним
        if c>mas[i]:                         # Якщо термін у правій частині масиву
            l = i+1                          # Зміна значення лівої межі
        else:                                # Якщо термін у лівій частині масиву
            p = i+1                          # Зміна значення правої межі
if per:                                      # Термін є у масиві?
    print ('Термін на позиції', i)           # Виведення позиції терміна
else:                                        # Термін у масиві відсутній
    print ('Термін відсутній')               # Повідомлення про відсутність терміна
```

Рис. 6.8. Програма бінарного пошуку елемента в одновимірному масиві

Один із варіантів виконання програми:

```
Увести термін: масив
Термін на позиції 4
```

Проблема оперативного пошуку даних набула особливої актуальності з появою Всесвітньої павутини.

6.2.3. Пошук максимального і мінімального елементів у масиві

Поміркуйте, як можна в одновимірному неупорядкованому масиві цілих чисел знайти максимальний або мінімальний елемент.

Пошук мінімального або максимального значення у масиві можна реалізувати різними способами.

Сутність одного з алгоритмів пошуку елемента з мінімальним значенням (пошук елемента з максимальним значенням принципово не відрізняється від алгоритму пошуку елемента з мінімальним значенням) така.

Крок 1	Спочатку мінімальним вважається елемент, розташований на першій позиції. Його значення порівнюється зі значенням другого елемента. Якщо значення другого елемента менше за значення першого, то далі меншим вважається другий елемент.
Крок 2	Потім значення меншого елемента порівнюється зі значенням третього елемента і так далі до останнього елемента. У результаті буде знайдено найменший елемент.

Розглянемо реалізацію алгоритму пошуку мінімального числа в масиві на прикладі 4.

Приклад 4. Виконаємо алгоритм пошуку мінімального числа в масиві: 42, 12, 55, 5, 37.
1-й цикл: min := 42; 12 < 42? Так — min := 12.
2-й цикл: 55 < 12? Ні.
3-й цикл: 5 < 12? Так — min := 5.
4-й цикл: 5 < 37? Так — результат: min = 5.

Програму реалізації алгоритму пошуку максимального і мінімального елементів у масиві зображено на рис. 6.9.

```
# Пошук максимального і мінімального елементів у масиві
mas = [22, 137, 31, 5, 41, 9]       # Створення масиву
n = len(mas)                        # Довжина масиву
max = mas[0]                        # Початкове значення максимального елемента
min = mas[0]                        # Початкове значення мінімального елемента
for i in range (n):                 # Циклічний перегляд елементів масиву
    if mas[i]>max:                  # Поточний елемент більший за максимальний?
        max = mas[i]                # Зміна значення максимального елемента
    if mas[i]>min:                  # Поточний елемент менший від мінімального?
        min = mas[i]                # Зміна значення мінімального елемента
print ('Максимальне значення =', max)
print ('Мінімальне значення =', min)
```

Рис. 6.9. Програма пошуку максимального і мінімального елементів у масиві

Результат виконання програми:

```
Максимальне значення = 137
Мінімальне значення = 5
```

6.2.4. Поняття про пошук із поверненням і тернарний пошук

Чи доводилося вам виконувати пошук варіантів виходу з лабіринту від його входу? Як у таких випадках ви діяли?

Пошук із поверненням (від англ. *backtracking*) — це загальний метод (або стратегія пошуку) розв'язування задачі, коли доводиться неодноразово повертатися до стану об'єкта (об'єктів), зафіксованого на попередньому кроці.

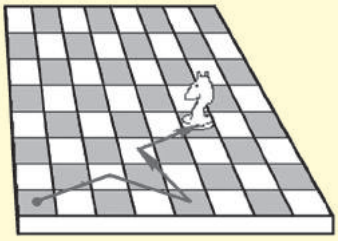
Класичним прикладом пошуку із поверненням є пошук у лабіринті всіх шляхів від входу до виходу з нього. Перелік усіх шляхів у лабіринті називають множиною всіх можливих рішень.

Прикладом пошуку з поверненням є також гра в шахи, коли для пошуку оптимального чергового ходу доводиться неодноразово повертатися до поточного стану білих і чорних фігур.

Пошук із поверненням може використовуватися для різних структур даних для розв'язування задач, у яких потрібно перерахувати всі можливі варіанти, наприклад, доставки вантажу з одної країни в іншу, або знайти способи отримання кредиту для будівництва готелю, або дати відповідь, чи існує такий спосіб вкладу, який задовольняє власні потреби клієнта тощо.

Алгоритм пошуку з поверненням зводиться до послідовного розширення часткового рішення. Якщо на черговому кроці таке розширення виконати не вдається, то здійснюється повернення на попередній крок і продовжується пошук.

Метод пошуку з поверненням можна зрозуміти на прикладі класичної задачі: на шахівниці n*n стоїть у клітинці (x, y) шаховий кінь. Слід знайти такий маршрут коня (який ходить згідно шахових правил), коли він обходить усю шахівницю, побувавши у кожній клітинці рівно один раз.



Алгоритм розв'язування задач методом пошуку з поверненням дозволяє знайти всі розв'язки для поставленого завдання, якщо вони існують.

Для прискорення методу намагаються так організувати обчислення, щоб якомога швидше можна було виявити варіанти, які не задовольняють умову (приклад 5).

Приклад 5.

У багатьох джерелах як класичний приклад алгоритму пошуку з поверненням описується задача про 8 ферзів, яку можна сформулювати так.

Необхідно на шаховій дошці розмістити 8 ферзів так, щоб жодний із них не був під боєм іншого.

1. Спочатку ставиться на шахову дошку один ферзь.
2. Далі ставляться інші так, щоб його не били вже поставлені ферзі.
3. Якщо на черговому кроці так поставити фігуру не можна, повертаються на крок назад і намагаються поставити раніше встановленого ферзя на інше місце.

За допомогою методу з поверненням можна отримати всі перестановки і комбінації даної множини.

Тернарний пошук в інформатиці застосовується для пошуку максимумів або мінімумів функції, яка на деякому відрізку спочатку постійно зростає, потім постійно спадає або спочатку спадає, а потім зростає.

Алгоритм тернарного пошуку можна реалізувати для пошуку заданого елемента в упорядкованому масиві, поділивши його на три приблизно рівні частини.

Елемент із заданим значенням можна порівняти з останнім елементом першої частини.

Якщо значення збігаються, то пошук завершується, якщо менше, то пошук продовжується в першій частині, інакше — він порівнюється з останнім елементом другої частини.

Далі виконуються дії, аналогічні описаним.

У світі існує близько 8500 мов програмування, але кількість мов, на яких пишуть програми професійні програмісти, не перевищує 10.

Запитання для перевірки знань

1. Які існують найпростіші методи пошуку даних у масиві?
2. Поясніть сутність алгоритму пошуку максимального числа в масиві.
3. Поясніть сутність послідовного методу пошуку даних у масиві.
4. Сформулюйте алгоритм обчислення кількості заданих чисел у масиві.
5. Поясніть сутність алгоритму бінарного пошуку даних у масиві.
6. Наведіть приклад послідовного пошуку даних у масиві.
7. Наведіть приклад бінарного пошуку даних у масиві.
8. Наведіть приклад тернарного пошуку даних у масиві.

Завдання для самостійного виконання

1. Дано масив рядків: 'байт', 'принтер', 'процесор', 'монітор'. Розробіть програму визначення позиції слова, що уводиться з клавіатури.
2. Дано масив цілих чисел: 13, 7, 6, 7, 9, 7, 6, 5. Розробіть програму обчислення кількості числа, значення якого вводиться з клавіатури.
3. Дано масив цілих чисел: 6, 8, 13, 17, 19, 30, 13, 8. Розробіть програму визначення всіх позицій, на яких знаходиться число, значення якого вводиться з клавіатури.
4. Дано масив рядків: 'блок', 'файл', 'біт', 'колонка', 'миша'. Розробіть програму пошуку максимального і мінімального значень елементів.
5. Розробіть програму створення масиву, що містить 10 випадкових чисел у діапазоні від 3 до 9 і визначення, чи є у ньому число 6.
6. Знайдіть в Інтернеті відомості про п'ять найвищих горних вершин в Україні. Розробіть програму визначення наявності у цьому переліку назви вершини, яка вводиться з клавіатури.

7. Обробка рядків

Рядок є одним з основних типів убудованих у мову Python об'єктів, які мають загальну назву «послідовність». Рядки можуть застосовуватися для подання символів, слів, фрагментів тексту, а також використовуватися для збереження двійкових значень байтів.

7.1. Основні відомості про рядки й операції над ними

З рядковим типом даних ви зустрічалися багаторазово. Які операції доводилося вам виконувати над рядками?

Основні операції над рядками:
- звернення до символу рядка;
- виділення фрагмента рядка;
- об'єднання двох рядків.

Рядок у мові Python є незмінним типом даних. Тобто змінювати символи в рядку не можна. Наприклад, спроба замінити в рядку "принтер" букву «и» на букву «о» призведе до появи повідомлення про синтаксичну помилку. Для змінення вмісту рядка застосовуються спеціальні засоби.

Основним типом рядків є str, призначений для роботи з текстовими даними у коді ASCII. Саме цей тип розглядатимемо далі. Мова Python містить значну кількість спеціальних символів, з яких найчастіше застосовуються \n (переведення рядка) і \r (повернення каретки).

Щоб символ виводився в тому самому вигляді, його слід екранувати: поставити перед ним слеш (\) (приклад 1).

Екранування символів — заміна в рядку керуючих символів на відповідні текстові підстановки. Ми уникаємо спеціальних символів, коли не хочемо, щоб вони виконували своє особливе призначення. Екранування відповідає на запитання: «Якщо ці символи такі особливі, то як мені їх використовувати у своєму тексті?»

Приклад 1

```
>>>print ("файл\nмиша") # Буде виведено два рядки
файл
миша
>>>print ("файл\\nмиша") # Буде виведено один рядок
файл\nмиша
```

Для роботи з рядками у мові Python є потужний набір засобів: операції, функції, методи й модулі. Розглянемо ті з них, що є основними та застосовуються найчастіше.

Для об'єднання двох рядків між ним слід розмістити слеш, або взяти їх у лапки, або використати конкатенацію всередині дужок (приклад 2).

У мові Python немає різниці між рядком в апострофах і рядком у лапках. Якщо сам рядок містить лапки, то його краще взяти в апострофи, а якщо містить апострофи — то в лапки.

Приклад 2

```
>>>"файл\ # Об'єднання рядків за допомогою слеш
миша"
'файлмиша'
>>>("файл» # Об'єднання рядків за допомогою круглих дужок
"миша")
'файлмиша'
>>>("файл"+ # Об'єднання рядків за допомогою операції
конкатенації
" миша")
'файл миша'
```

Якщо після слеша немає символа, який разом з ним інтерпретується як спеціальний, то слеш зберігається в рядку:

```
>>>print ("Наведемо перелік \спеціальних символів")
Наведемо перелік \спеціальних символів
```

Розглянемо основні операції над рядками.

- Звернення до символу рядка потребує зазначення імені рядка та у квадратних дужках його індекса (нумерація починається з нуля):

```
>>>slp = "вінчестер"
>>>slp[2], slp[8] # Виведення 2-го і 8-го символів
('н', 'р')
```

- Виділення фрагмента рядка виконується за допомогою операції:

ім'я рядка [початок: кінець: крок]

Як бачимо, усі параметри тут є необов'язковими. За замовчуванням вони мають такі значення: початок — 0, кінець — номер індекса останнього символу, крок, що дорівнює одиниці:

```
>>>a_1 = "процесор"
>>>a_1 [:] # Виділяється увесь рядок
процесор
>>>a_1 [3:6] # Виділяється фрагмент цес
цес
```

- Об'єднання двох рядків реалізується за допомогою оператора конкатенації (+):

```
>>>print ("клавіа"+"тура")
клавіатура
>>>a = "Системний"
>>>print (a+" блок")
Системний блок
```

Екрановані послідовності, так звані escape-послідовності, можуть складатися з одного або декількох символів після зворотної скісної риски.

Якщо в рядку не існує символа з указаним індексом, буде видано повідомлення про помилку. Як відомо, рядки є незмінними типами даних, тому змінити символ за індексом неможливо.

Запитання для перевірки знань

1. Який основний тип мають рядки?
2. Як можна звернутися до символу рядка?
3. Наведіть приклад об'єднання двох рядків в один.
4. Яке призначення мають символи \n?
5. Поясніть, для чого в рядках використовують слеш (\).
6. Як у рядку можна виділити фрагмент?

Завдання для самостійного виконання

1. Об'єднайте рядки "Створення анімації методом" і "трансформації руху" в один рядок.
2. Визначте символ, розташований на 13-й позиції рядка "Операція конкатенації".
3. У рядку "Змінити символ за індексом неможливо" виділіть фрагмент із 9-ї до 26-ї позиції.
4. Об'єднайте рядки "Як у рядку можна" і "виділити його фрагмент" в один рядок, а потім перетворіть його у рядок "у рядку можна виділити його фрагмент".
5. Перетворіть рядок "Усі параметри тут є необов'язковими" у рядок "Параметри є обов'язковими".

7.2. Функції і методи опрацювання рядків

Яка, на вашу думку, потреба у використанні функцій і методів опрацювання рядків? Чому не можна обійтися тільки операціями над рядками?

Ви вже ознайомилися з операціями для роботи з рядками. Для опрацювання рядків у мові Python існує значна кількість функцій, основні з яких наведено в табл. 7.1.

Таблиця 7.1. Основні функції опрацювання рядків

Функція	Призначення
len()	Визначення кількості символів у рядку: >>>len("монітор") **7**
str()	Перетворення послідовності у рядок. Якщо необхідно з'єднати рядок з іншим типом даних (числом, списком та іншими), то його слід перетворити в рядок за допомогою функції str(). Найпростіша структура функції така: str(об'єкт): >>>str(), str([5, 6, 7]), str((8, 9)), str({"a" : 2}) # Перетворення послідовностей **('', '[5, 6, 7]', '(8, 9)', "{'a' : 2}")** Рядки також можна повторювати, виконувати перевірку на входження і невходження одного рядка в інший. Рядки можна форматувати багатьма способами за допомогою значної кількості операцій форматування і методу format()
chr(код символу)	Повертає символ зазначеного коду: >>>print (chr(1065)) **Щ**
ord(символ)	Повертає символ: >>>print (ord("Щ")) **1065**

До основних методів роботи з рядками можна віднести методи, які наведено в табл. 7.2.

Таблиця 7.2. Основні методи роботи з рядками

Метод	Опис
join ()	Перетворює рядкові елементи послідовності в один рядок. Він має таку структуру: <відокремлювач>.join (послідовність): >>>" ".join (("Рядкові", "елементи", "послідовності")) '**Рядкові елементи послідовності**'
upper ()	Замінює у рядку всі малі букви великими: >>>print ("комп'ютер".upper ()) **КОМП'ЮТЕР**
lower ()	Замінює у рядку великі букви малими: >>>print ("КОМП'ЮТЕР".lower ()) **комп'ютер**
capitalize ()	Замінює першу букву рядка великою: >>>print ("комп'ютер".capitalize ()) **Комп'ютер**

Для пошуку і заміни в рядку застосовують понад десять методів, основні з яких наведено в табл. 7.3.

Таблиця 7.3. Основні методи пошуку і заміни в рядку

Метод	Опис
find ()	Здійснює пошук підрядка в рядку. Якщо підрядок знайдено, повертається номер позиції, після якої починається підрядок. Інакше — повертається значення –1. Формат методу: <рядок>.find (підрядок, [початок [, кінець]]) Якщо параметри початок і кінець не вказані, здійснюється пошук від самого початку рядка до його кінця: >>>sl = "пошук підрядка в рядку" >>>sl.find ("шук"), sl.find ("підряд"), sl.find ("кут") **(2, 6, –1)**
index ()	Відрізняється від методу find () лише тим, що у випадку, коли підрядок у рядку відсутній, генерується виняток ValueError
rfind ()	Відрізняється від методу find () лише тим, що повертається номер позиції рядка, на якій закінчується підрядок
count ()	Визначає кількість входжень підрядка в рядок: >>>sl = "пошук підрядка в рядку" >>>sl.count ("ряд"), sl.count ("шук"), sl.count ("мас") **(2, 1, 0)**
replace ()	Виконує заміну всіх входжень заданого підрядка в рядку на новий підрядок і повертає новий рядок. Формат методу: <рядок>.replace (<підрядок заміни>, <новий підрядок>) >>>print ("старий комп'ютер".replace ("старий", "новий")) **новий комп'ютер**

Мова Python реалізує також складний пошук і заміну. Для цього існує низка методів, які містяться у модулі re (тут не розглядається).

Запитання для перевірки знань

1. Для чого призначена функція len ()?
2. Яке призначення має функція chr ()?
3. Для чого призначений метод find ()?
4. Поясніть сутність методу replace ().
5. Для чого призначений метод count ()?
6. Поясніть сутність методу upper ().
7. Для чого призначений метод join ()?
8. Порівняйте методи find () і rfind ().

Завдання для самостійного виконання

1. Дано рядки: "Операційна" і "система". Визначте довжину кожного рядка, об'єднайте їх в один рядок і визначте його загальну довжину.
2. Дано чотири рядки: "Великі річки України:", "Дніпро", "Дністер", "Десна". Визначте довжину кожного рядка, об'єднайте їх в один і визначте довжину нового рядка.
3. Дано рядок "Рим — столиця Італії, Київ — столиця України". Визначте код букви **м** та кількість входжень підрядка в рядок столиця.
4. Дано рядок "microsoft Word". Замініть першу букву великою буквою **М**, визначте кількість букв **о** в рядку і код букви на п'ятій позиції.
5. Дано рядок "Інструкція для користувача". Замініть слово **користувача** словом **учня** і визначте позицію, з якої починається підрядок **для**.

7.3. Приклади програм обробки рядків

Наведіть приклади опрацювання рядків, з якими вам доводилося зустрічатися в процесі вивчення інших предметів, наприклад географії, фізики.

Приклад 1. Дано два рядки: "Рядок є незмінним" і "типом даних". На рис. 7.1 зображено програму об'єднання цих рядків, визначення довжини створеного рядка, виділення підрядка і заміни одного підрядка іншим.

```
a = "Рядок є незмінним"          # Перший рядок
b = "типом даних"                # Другий рядок
c = a+" " +b                     # Об'єднання рядків
print (c)                        # Новий рядок
print (len(c))                   # Довжина нового рядка
print (c[8:17])                  # Виділення підрядка
                                 # Заміна одного підрядка іншим
print (c.replace ("типом даних", "об'єктом"))
print (c)                        # Створений рядок не змінився
```

Рис. 7.1. Програма об'єднання рядків

Останню інструкцію print (c) спеціально додано до програми для того, щоб звернути увагу на те, що метод replace не змінює сам рядок, а лише повертає внесені зміни. Результат виконання програми є таким:

```
Рядок є незмінним типом даних
29
незмінним
Рядок є незмінним об'єктом
Рядок є незмінним типом даних
```

Приклад 2. Дано рядки: ("кількість", "входжень") і ("підрядка", "в рядок"). Розробити програму об'єднання їх в один рядок і перетворення рядкових елементів в один рядок, визначення кількості входжень у рядок слова підрядка і коду букви к.

Програму реалізації цього завдання зображено на рис. 7.2.

```
a = ("кількість", "входжень")  # Перший рядок
b = ("підрядка", "в рядок")    # Другий рядок
c = a+b                        # Об'єднання рядків
print (c)                      # Виведення рядка
                               # Перетворення рядкових
                                 елементів в один рядок

print (" ".join (c))
print (c.count ("підрядка"))    # Кількість входжень
print (ord("к"))                # Код букви к
```

Рис. 7.2. Програма перетворення рядкових елементів в один рядок

Оскільки початкові рядки складаються з рядкових елементів, то підрядком у методі count може бути повністю один із них. У цьому випадку не можна розглядати як підрядок частину рядкового елемента, інакше буде видано повідомлення про синтаксичну помилку.

Результат виконання програми:

```
('кількість', 'входжень', 'підрядка', 'в рядок')
кількість входжень підрядка в рядок
1
107
```

Приклад 3. Дано рядок: "рядок є незмінним типом даних". Замінити першу букву рядка на велику, визначити код букви на 8-й позиції, кількість букв м у рядку, позицію, з якої починається слово «типом», і кількість букв у слові «незмінним».

Програму реалізації завдання зображено на рис. 7.3.

```
a = "рядок є незмінним типом даних"
print (a.capitalize ())          # Перша буква рядка замінюється на велику
print (ord(a[8]))                # Код букви на 8-й позиції
print (a.count ("м"))            # Кількість букв м у рядку
print (a.find ("типом"))         # Позиція, з якої починається слово "типом"
print (len("незмінним"))         # Довжина слова "незмінним"
```

Рис. 7.3. Програма опрацювання одного рядка

Результат виконання програми:

```
Рядок є незмінним типом даних
1085
3
18
9
```

Завдання для самостійного виконання

1. Складіть програму, за допомогою якої рядок "Львів — гарне місто для екскурсій" перетворіть на рядок "Львів — гарне місто для екскурсій та подорожей".
2. Складіть програму, за допомогою якої рядки "Річка Сіверський Донець" і "впадає у річку Дон" об'єднуються в один рядок і визначається його довжина.
3. Складіть програму, за допомогою якої рядки "Найвища вершина", "України Говерла" і "належить до", "Карпатських гір" об'єднуються в один рядок і рядкові елементи перетворюються в один рядок.
4. Складіть програму об'єднання рядків "Файлова система" та "операційної системи Windows". Визначте довжину нового рядка і кількість входжень слова "система" в цей рядок.
5. Складіть програму, за допомогою якої у рядку "програма медіапрогравач Windows поєднує багато функцій" перша буква замінюється великою буквою, визначається код букви на 10-й позиції і кількість букв «о» в рядку.
6. Складіть програму, за допомогою якої у рядку "Київський ботанічний сад" визначається довжина слова "ботанічний", позиція, з якої починається слово "сад", кількість букв н у слові "ботанічний" і код букви на 5-й позиції.

8. Графи

8.1. Основні поняття і терміни теорії графів

Чи доводилося вам розв'язувати задачі, в яких розглядається деяка сукупність об'єктів, між якими заданий певний зв'язок?

З графами нам неодноразово доводилося зустрічатися в повсякденному житті. Існує багато задач, які можна розв'язати за допомогою графів. Це задачі пошуку найкоротшого шляху від одного населеного пункту до іншого, якщо відома карта доріг, задача маршрутизації трафіку, якщо відомий час проходження інформації від одного сервера до іншого, та багато інших. Розглянемо приклад.

Наприклад, графом можна назвати схеми метрополітену, автошляхів між містами регіону, схему комп'ютерної мережі.

Приклад. В 11 класі проводиться шаховий турнір, у якому беруть участь Катерина, Марія, Володимир, Олександр і Петро. На даний момент грали: Катерина з Володимиром і Марією, Володимир з Катериною, Марією і Петром, Марія з Володимиром, Катериною і Петром, Катерина з Олександром, Олександр з Марією і Петром, Петро з Володимиром і Олександром. Стан гри учасників відображений на схемі (рис. 8.1).

На схемі кожний учасник позначений першою буквою свого імені, а суцільними лініями показано ігри, що вже відбулися між учасниками турніру, пунктирними — ігри, які ще не відбулися.

З рис. 8.1 видно, що відбулося 6 ігор і 3 мають відбутися.

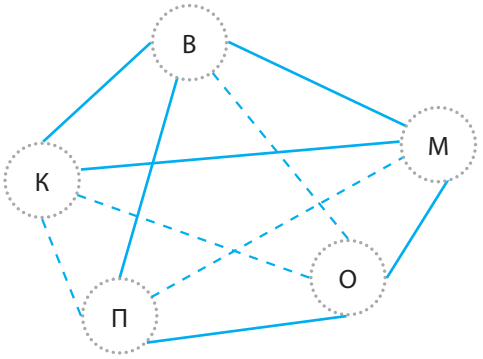


Рис. 8.1. Схематичне зображення стану гри в шахи

Сукупність об'єктів, а також набір всіх зв'язків між ними називають графом. У цьому випадку об'єкти називають *вершинами* графу, а зв'язки між ним — *ребрами* графу.

Графи зазвичай подаються у вигляді геометричних фігур так, що вершини графу зображуються точками, а ребра — лініями, що з'єднують ці точки.

Вершина може мати ідентифікатор — цифру або букву.

Ребра графа можуть мати напрямок, який позначається стрілкою. Таке орієнтоване ребро називається *дугою*. Наприклад, вулиці міста з одностороннім рухом можна позначати дугою.

Якщо всі ребра в графі мають напрямок (тобто є дугами), то граф називається *орієнтованим*, якщо всі ребра не мають напрямку, граф називається *неорієнтованим*. Якщо граф має і ребра, і дуги, він називається *змішаним*.

На рис. 8.2 наведено приклад найпростішого графу з ребрами і дугами. Тут A, B, C — вершини графу, AB — дуга, AC і BC — ребра. Вершина C з'єднана ребром сама із собою. Такі ребра називають *петлею*.

Дві вершини, з'єднані ребром або дугою, називають *суміжними*. Суміжними на рис. 8.2 є вершини A і B, A і C, B і C.

Граф, у якому будь-які дві вершини з'єднані ребрами, називають *повним*. Приклад повного графу наведено на рис. 8.3.

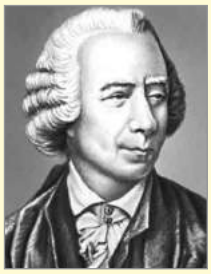
Степенем вершини називають число ребер, яким належить ця вершина. Наприклад, на рис. 8.3 кожна вершина має степінь 3. Залежно від парності степеня розрізняють парні та непарні вершини.

Шляхом у графі називають послідовність його ребер, які зустрічаються при переміщенні з однієї вершини в іншу. Першу вершину називають *стартовою* (*початком шляху*), а останню — *кінцевою* (*кінцем шляху*). Наприклад, на рис. 8.4 із вершини 1 у вершину 4 є такі шляхи: 1–4, 1–5–4 і 1–2–3–4.

Довжиною шляху називають кількість ребер, що входять у цей шлях. У розглянутому прикладі довжина другого шляху дорівнює 2, а третього — 3. Жодне ребро на шляху не повинно зустрічатися більше одного разу.

Неорієнтований граф називається *зв'язаним*, якщо з будь-якої його вершини можна потрапити в будь-яку іншу вершину. Тобто між будь-якою парою вершин цього графу існує як мінімум один шлях.

Повний граф завжди є зв'язаним. Але не кожний зв'язаний граф є повним. Наприклад, граф, зображений на рис. 8.4, є зв'язаним, але не є повним.



Родоначальником теорії графів вважається Леонард Ейлер. У 1736 році Ейлер запропонував схему семи Кенігсберзьких мостів, яка пізніше стала однією з класичних задач теорії графів.

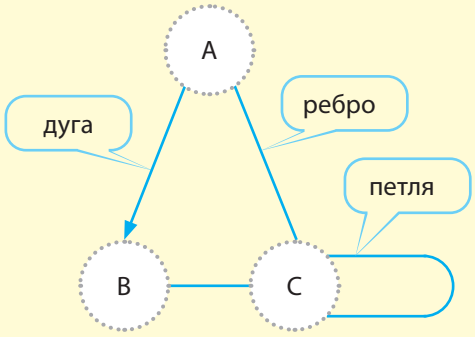


Рис. 8.2. Найпростіший граф із ребрами та дугами

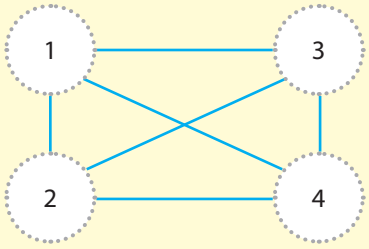


Рис. 8.3. Приклад повного графу

Вершина графу, яка належить лише одному ребру, називається *висячою*. На рис. 8.5 висячою є вершина 1.

Вершина 0 не з'єднана із жодною іншою. Такі вершини називають *ізольованими*.

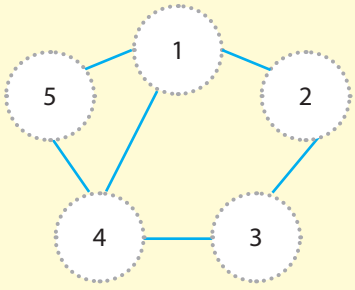


Рис. 8.4. Приклад зв'язаного, але неповного графу

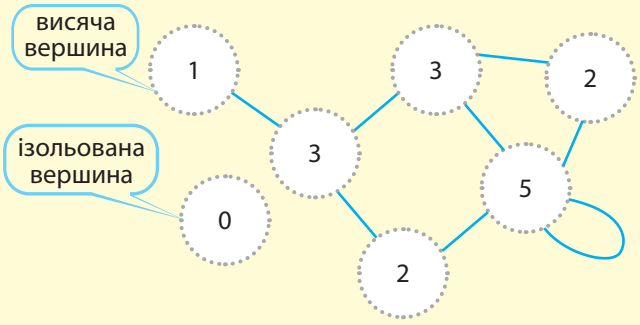


Рис. 8.5. Граф із висячою й ізольованою вершинами та петлею

Циклом називають шлях з однієї вершини графу в ту саму вершину. Циклів у графі для кожної вершини може бути декілька.

Розглянемо приклади циклів для графу, зображеного на рис. 8.6. Для вершини 2 у цьому графі є такі цикли:

2–1–5–2;
2–5–1–2;
2–3–4–5–2;
2–5–4–3–2;
2–1–5–4–3–2;
2–3–4–5–1–2.

Довжиною циклу називають кількість ребер у циклі.

Графи, які можна намалювати, не відриваючи олівця від паперу, називають ейлеровими. Щоб визначити, чи є граф ейлеровим, треба визначити степені кожної вершини.

Якщо в графі більше двох непарних вершин, то намалювати фігуру неможливо.

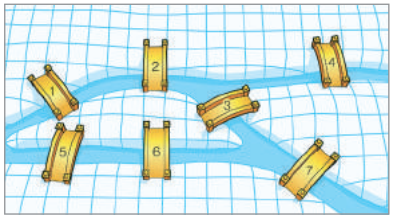
Якщо всі вершини графу парні, то намалювати фігуру можливо, і починати можна з будь-якої вершини.

Якщо в графі дві непарні вершини, то намалювати фігуру можна, але тільки починати необхідно в одній з цих двох непарних вершин, а закінчувати в іншій непарній вершині.

Граф, який не має жодного циклу, називають *деревом*. Приклад такого графу зображено на рис. 8.7.

Ребро називають *мостом*, якщо воно є єдиним шляхом, що зв'язує дві вершини. Прикладом моста на графі, зображеному на рис. 8.7, є ребро 1–2.

Класична задача теорії графів має таку історію. У 1736 році в місті Кенігсберзі було два острови, з'єднані 7 мостами з берегами річки Преголь і один з одним. Сутність задачі: здійснити прогулянку містом таким чином, щоб, пройшовши один раз по кожному мосту, повернутися в те саме місце, звідки починалася прогулянка.



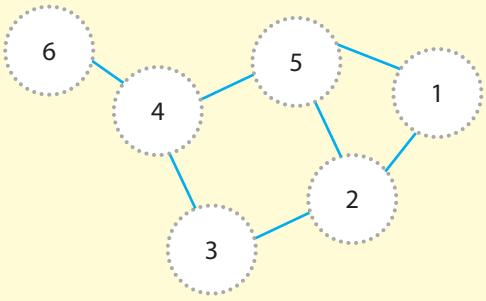


Рис. 8.6. Граф із циклами

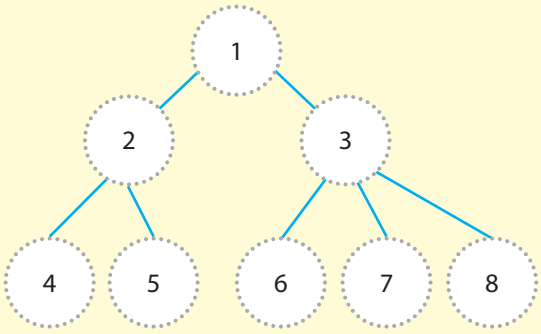


Рис. 8.7. Дерево (як вид графу)

Ребра графу можуть мати вагу, яка позначається числами на ребрах самого графу.

Граф (орієнтований і неорієнтований), усі ребра якого мають вагу, називають зваженим (рис. 8.8, 8.9).

Приклад неорієнтованого зваженого графу зображено на рис. 8.9. Тут вага ребра A–B дорівнює 3, а ребра B–C — 7.

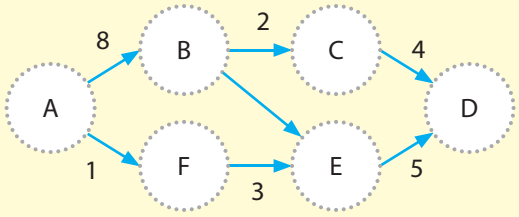


Рис. 8.8. Орієнтований зважений граф

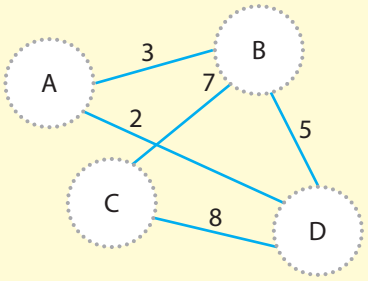


Рис. 8.9. Неорієнтований зважений граф

Запитання для перевірки знань

1. Наведіть визначення графу.
2. Як позначають вершини графу?
3. Яка різниця між дугами і ребрами графу?
4. Які графи називають повними?
5. Наведіть приклад графу з ребрами і дугами.
6. Які вершини графу називають суміжними?
7. Які графи називають орієнтованими; неорієнтованими; змішаними?
8. Поясніть сутність степеня вершини.
9. Які графи називають зв'язаними?
10. Поясніть сутність циклу в графі.

Завдання для самостійного виконання

1. Накресліть повний граф із п'ятьма вершинами.
2. На графі, зображеному на рис. 8.9, визначте всі можливі довжини шляхів із вершини C у вершину A.
3. У неділю учні 11 класу домовилися відвідати ботанічний сад. Час і місце зустрічі доручили визначити Вові, який мав повідомити їх Юлі й Миколі. Юля повинна повідомити їх Тані й Віті, Вітя — Наталі й Ігорю, Микола — Олі й Петру, Петро — Олені й Івану. Оформте схему оповіщення у вигляді графу.
4. Розробіть граф зв'язків основних пристроїв вашого домашнього або шкільного комп'ютера.
5. Накресліть зважений граф доріг державного значення між чотирма обласними центрами України, вибраними за вказівкою вчителя. На ребрах вкажіть довжину доріг.
6. Розробіть граф для станцій метро Києва: Поштова площа, Майдан Незалежності, площа Льва Толстого, Університетська, Театральна, Хрещатик, Арсенальна з усіма можливими переходами між цими станціями, користуючись рисунком.

8.2. Способи подання графів у комп'ютері

Пригадайте, як ви подавали графи під час вивчення математики та інших навчальних дисциплін.

Існують декілька способів подання графів у комп'ютері, розглянемо два з них: за допомогою матриць суміжності та списку суміжних вершин.

Подання графів за допомогою матриць суміжності. Матриця суміжності для графу з n вершинами подається двовимірним масивом розмірністю n*n.

Якщо вершина i *неорієнтованого незваженого графу* має ребро з вершиною j, то елемент масиву mas[i, j] набуває значення 1, інакше — 0.

Розробимо матрицю суміжності для неорієнтованого графу, зображеного на рис. 8.10.

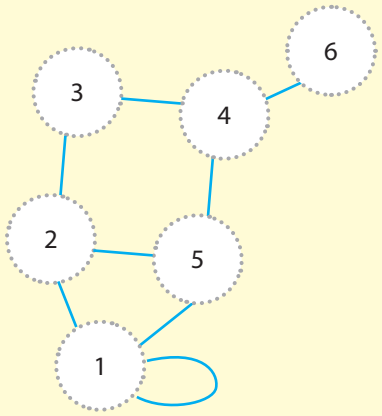


Рис. 8.10. Неорієнтований незважений граф (із петлею)

Матрицю суміжності для неорієнтованого незваженого графу наведено на рис. 8.11.

Вершини	Вершини					
	1	2	3	4	5	6
1	1	1	0	0	1	0
2	1	0	1	0	1	0
3	0	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	0
6	0	0	0	1	0	0

Рис. 8.11. Матриця суміжності для неорієнтованого незваженого графу з петлею

Для орієнтованого незваженого графу елементи матриці суміжності набувають значення 1, якщо відповідне ребро виходить з вершини, інакше — значення 0.

На рис. 8.12 ребро виходить із вершини 1 у вершини 2 і 3, тому елементи mas[1, 2] і mas[1, 3] набувають значення 1, а елемент mas[2, 1] — значення 0.

Матрицю суміжності для *орієнтованого незваженого графу* (див. рис. 8.12) наведено на рис. 8.13.

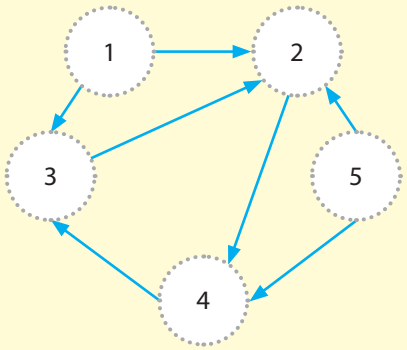


Рис. 8.12. Орієнтований незважений граф

Вершини	Вершини				
	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	0
5	0	1	0	1	0

Рис. 8.13. Матриця суміжності для орієнтованого незваженого графу

Розглянемо подання *орієнтованого зваженого графу* (рис. 8.14). Матрицю суміжності для цього графу наведено в таблиці на рис. 8.15. Як бачимо, елементи матриці набувають значень ваги відповідних ребер.

Вершини	Вершини				
	1	2	3	4	5
1	0	8	0	15	3
2	0	0	5	0	0
3	0	0	0	0	17
4	0	0	13	0	9
5	0	0	0	0	0

Рис. 8.15. Матриця суміжності орієнтованого зваженого графу

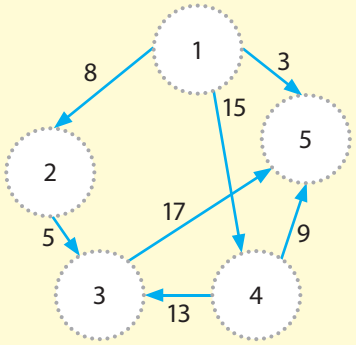


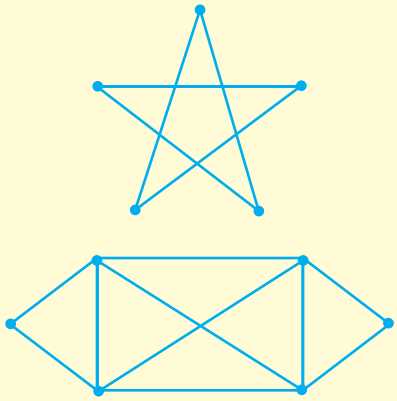
Рис. 8.14. Орієнтований зважений граф

Подання графів за допомогою списку суміжних вершин. Для кожної вершини складається список суміжних вершин. Наприклад, для графу (див. рис. 8.12) список суміжних вершин можна записати так:

1 – 2 – 3
2 – 4
3 – 2
4 – 3
5 – 2 – 4

Для зваженого графу біля суміжної вершини вказується вага ребра. Наприклад, для графу (див. рис. 8.14) список можна записати так:

1 – $2_{(8)}$ – $4_{(15)}$ – $5_{(3)}$
2 – $3_{(5)}$
3 – $5_{(17)}$
4 – $3_{(13)}$ – $5_{(9)}$
5 –



Приклади ейлерового графу

Запитання для перевірки знань

1. Які способи подання графів у комп'ютері?
2. Яка структура даних використовується в матриці суміжності?
3. Поясніть порядок створення матриці суміжності для неорієнтованого незваженого графу.
4. Поясніть порядок подання графу списком суміжних вершин.
5. Як створюється матриця суміжності для орієнтованого зваженого графу?
6. Наведіть приклад подання зваженого графу списком суміжних вершин.

Завдання для самостійного виконання

1. Складіть матрицю суміжності для графу, зображеного на: а) рис. 8.4; б) рис. 8.6; в) рис. 8.8; г) рис. 8.9.
2. Знайдіть в Інтернеті відомості про населені пункти, розташовані у вашій області, з кількістю населення понад 50 000 осіб і довжину автомобільних шляхів між ними. Розробіть граф автомобільних шляхів між цими населеними пунктами та список суміжних вершин.

8.3. Пошук у глибину та ширину

Знайдіть в Інтернеті схему метрополітену Києва, яка є різновидом графу. Сформулюйте алгоритм пошуку маршруту з однієї станції метро до іншої.

У процесі опрацювання графів часто доводиться виконувати обхід усіх його вершин. Існують декілька алгоритмів перегляду вершин графу, розглянемо два з них: алгоритми пошуку у глибину та пошуку в ширину.

Алгоритм пошуку в глибину. Для реалізації алгоритму пошуку в глибину використовують дві структури даних: стек для запам'ятовування ще не опрацьованих вершин і список для запам'ятовування вже опрацьованих вершин.

Розглянемо сутність алгоритму пошуку в глибину.

Крок 1	Вибирають стартову вершину, наприклад V.
Крок 2	Стартову вершину включають у список опрацьованих (переглянутих) вершин.
Крок 3	У стек включають усі вершини, суміжні зі стартовою.
Крок 4	Створюють цикл за умовою появи порожнього стека, усередині якого: 1) вибирають зі стека чергову вершину; 2) перевіряють у списку, чи опрацьована ця вершина. Якщо опрацьована, то зі стека вибирають чергову вершину. Якщо вершина ще не опрацьована, то вибирають її і розміщують у список опрацьованих вершин; 3) переглядають весь список суміжних із нею вершин і розміщують у стек ще не опрацьовані вершини.

Наведемо пояснення до цього алгоритму.

Якщо стартовою є вершина V, то вибираємо вершину U, що суміжна з вершиною V.

На наступному кроці для вершини U вибираємо суміжну з нею вершину.

Якщо на черговому кроці розглядається вершина Q і немає вершин, суміжних із нею, і таких, що не переглядалися раніше, то повертаємося з вершини Q до тієї вершини, з якої потрапили у вершину Q і т. д. Якщо це вершина, з якої починався перегляд, тобто вершина V, і непереглянутих суміжних з нею вершин немає, то перегляд завершений.

Розглянемо порядок перегляду вершин графу в глибину на прикладі 1 неорієнтованого незваженого графу (рис. 8.16, а).

Приклад 1. Перегляд вершин можна розпочати з будь-якої вершини, наприклад із вершини 3. Із неї можна рухатися і у вершину 2, і у вершину 7.

Потім перейдемо до вершини 2, з якої рухаємося до вершини 1. Але у вершини 1 відсутня суміжна для неї вершина, яка ще не розглядалася.

Повертаємося у вершину 2 і рухаємося до вершини 4. У вершини 4 також відсутні суміжні вершини, які ще не розглядалися.

Повертаємося ще раз у вершину 2 і рухаємося у вершину 5. З неї рухаємося у вершину 6, потім у вершину 7.

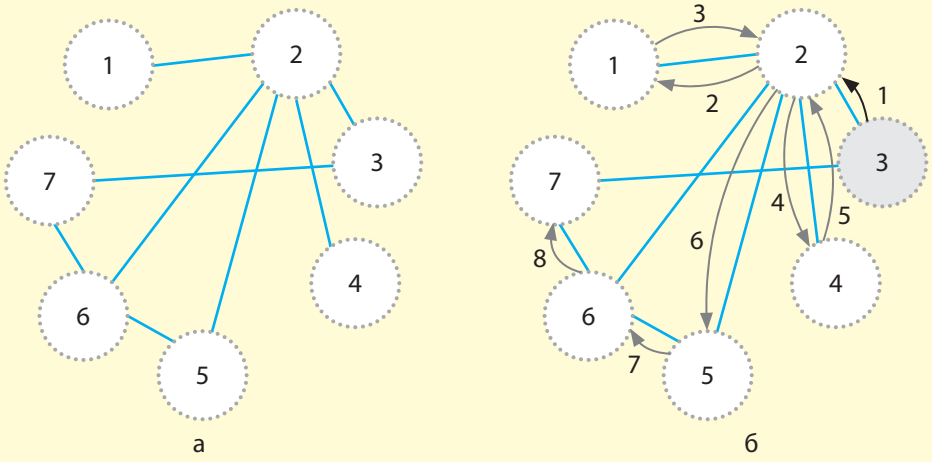


Рис. 8.16. Неорієнтований незважений граф (*а*), неорієнтований незважений граф, стрілкою позначено напрям руху, цифрою — номер кроку (*б*)

Приклад 2. Список [[1, 2], [0, 2], [0, 1]] вказує, що вершина 0 має зв'язок із вершинами 1 і 2, вершина 1 — з вершинами 0 і 2, а вершина 2 — з вершинами 0 і 1.

Для реалізації алгоритму пошуку мовою Python нумерацію вершин графу доцільно починати з 0. Суміжні вершини графу можна подавати списками (приклад 2).

Приклад 3. Нехай дано граф із шістьма вершинами, які мають між собою такі зв'язки: [[4, 5], [5], [3, 4], [2, 4], [0, 2, 3], [0, 1]].

Програму обходу вершин графу в глибину, *починаючи з нульової вершини*, зображено на рис. 8.17.

```
# Пошук у глибину
graph = [[4, 5], [5], [3, 4], [2, 4], [0, 2, 3], [0, 1]]     # Список суміжності вершин
stan = [False for i in range (len(graph))]          # Список початкового стану вершин
print (stan)                                # Виведення початкового стану вершин
print ('Порядок обходу вершин')             # Повідомлення
def func (v):                               # Заголовок функції
    print (v, end =" ")                     # Результат порядку обходу вершин
    stan[v] = True                          # Значення стану поточної вершини
    for vartex in graph[v]:                 # Цикл опрацювання кожної вершини
        if not stan[vartex]:                # Якщо був обхід цієї вершини
            func (vartex)                   # Повернення до функції
for c in range (len(graph)):                # Циклічне звернення до функції
    if not stan[c]:                         # Якщо значення поточної вершини True
        func (c)                            # Звернення до функції
```

Рис. 8.17. Програма обходу вершин графу в глибину

Результат виконання програми:

```
[False, False, False, False, False, False]
Порядок обходу вершин
0 4 2 3 5 1
```

Алгоритм пошуку в ширину. Спочатку опрацьовуються всі вершини, суміжні з поточною, а потім — «нащадки». Замість стека для збереження ще не опрацьованих вершин використовується *черга*. Сутність алгоритму пошуку в ширину така.

Крок 1	Обирається стартова вершина, включається у список опрацьованих вершин.
Крок 2	У черзі запам'ятовуються всі вершини, суміжні зі стартовою.
Крок 3	Створюється цикл за умови порожньої черги, усередині якого: 1) із черги вибирається чергова вершина; 2) у списку перевіряється, чи опрацьована ця вершина. Якщо вона вже опрацьована, то із черги вибирається наступна вершина. Якщо вершина не опрацьована, то необхідно опрацювати її і помістити у список опрацьованих; 3) переглядається весь список суміжних із нею вершин і поміщаються в чергу ще не опрацьовані вершини.

Наведемо пояснення до цього алгоритму.

Наприклад, якщо стартовою є вершина A0, то опрацьовуються всі суміжні з нею вершини A1, A2, ..., AK, далі — суміжні вершини для кожної із вершин A1, A2, ..., AK і т. д. Будуть перебрані всі вершини, і пошук завершиться.

Розглянемо процес пошуку в ширину на прикладі графу (рис. 8.16, *а*). Порядок обходу вершин із вершини 3: з вершини 3 переглядаються вершини 2 і 7; з вершини 2 — вершини 1, 4, 5, 6 (рис. 8.16, *в*).

Приклад 4. Нехай дано граф із шістьма вершинами, які мають такі зв'язки між собою: [[1, 3], [0, 3, 4, 5], [4, 5], [0, 1, 5], [1, 2], [1, 2, 3]].

На рис. 8.18 зображено програму обходу вершин графу в ширину, починаючи з нульової вершини.

```
# Пошук у ширину
graph = [[1, 3], [0, 3, 4, 5], [4, 5], [0, 1, 5], [1, 2], [1, 2, 3]]      # Список
                                    # суміжності вершин
pozhat = [-1]*len(graph)            # Список початкового перегляду вершин
print ('Початковий стан', pozhat)   # Виведення списку початкового стану вершин
def func (s):                       # Функція з параметром
    global pozhat                   # Глобальна змінна pozhat
    pozhat[s] = 0                   # Початкова вершина
    zherg = [s]                     # Додавання початкової вершини у чергу
    print ('Динаміка зміни стану перегляду вершин')
    while zherg:                    # Доти, доки черга не порожня
        print (pozhat)
        v = zherg.pop(0)            # Обираємо вершину
        for i in graph[v]:          # Цикл обходу із вершини v
            if pozhat[m] is -1:     # Перевірка, чи переглядали вершину
                zherg.append (m)    # Обираємо вершину
                pozhat[m] = pozhat [v]+1    # Визначення номера перегляду вершини
for i in range (len(graph)):        # Циклічне звернення до функції
    if pozhat[i] is -1:             # Якщо є елемент зв'язності
        func (i)                    # Звернення до функції
    print ('Вершина', i, ', номер обходу', pozhat[i])      # Номери перегляду вершин
```

Рис. 8.18. Програма пошуку в ширину

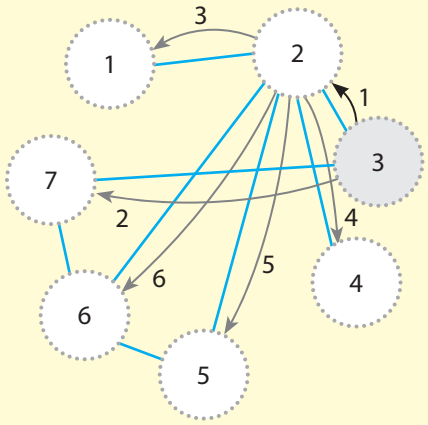


Рис. 8.16, *в*. Неорієнтований незважений граф, стрілкою позначено напрям руху, цифрою — номер кроку

Результат виконання програми наведено до прикладу 4:

```
Початковий стан [-1, -1, -1, -1, -1, -1]
Динаміка зміни стану перегляду вершин
[0, -1, -1, -1, -1, -1]
[0, -1, -1, 1, -1, -1]
[0, 1, -1, 1, 2, 2]
[0, 1, -1, 1, 2, 2]
[0, 1, 3, 1, 2, 2]
[0, 1, 3, 1, 2, 2]
Вершина   0, номер обходу 0
Вершина   1, номер обходу 1
Вершина   2, номер обходу 3
Вершина   3, номер обходу 1
Вершина   4, номер обходу 2
Вершина   5, номер обходу 2
```

Запитання для перевірки знань

1. Чому для реалізації алгоритмів пошуку в ширину і довжину мовою Python доцільно позначати вершини графу цифрами, починаючи з 0?
2. Поясніть послідовність перегляду в глибину вершин графу на рис. 8.16, *а*, починаючи з вершини 5.
3. Поясніть послідовність перегляду в ширину вершин графу на рис. 8.16, *а*, починаючи з вершини 7.
4. Сформулюйте сутність алгоритму пошуку в глибину.
5. Сформулюйте сутність алгоритму пошуку в ширину.

Завдання для самостійного виконання

1. Список суміжних вершин графу має такий зміст: [[1, 3], [0, 2, 3], [1, 3], [0, 1, 2]]. Розробіть граф, який відповідає цьому списку.
2. Модифікуйте програму, зображену на рис. 8.18, так, щоб обхід починався з вершини 2. Виконайте програму і доведіть правильність отриманих результатів.
3. Для списку суміжності вершин, який використовується в програмі (рис. 8.16, *а*), розробіть відповідний граф.
4. Модифікуйте програму (рис. 8.17) так, щоб обхід починався з вершини 3.
5. Виконайте програму, зображену на рис. 8.18, для нового списку суміжних вершин, розробленого самостійно. Доведіть правильність отриманих результатів.
6. Розробіть граф залізничних шляхів між шістьма населеними пунктами України, які виберіть на власний розсуд або за вказівкою вчителя. Створіть для цього графу список суміжних вершин і використайте його у програмі, зображеній на рис. 8.17. Виконайте програму і доведіть правильність отриманих результатів.

8.4. Визначення найкоротшого шляху в графі

З однієї станції метро можна потрапити до іншої станції різними шляхами. Наведіть приклад пошуку найкоротшого шляху у відомому вам метро.

У процесі пошуку найкоротшого шляху у графі між двома вершинами відшукується шлях, для якого сума ваг його ребер є мінімальною серед усіх інших шляхів. Така задача може виконуватися для неорієнтованого й орієнтованого графів.

Задача пошуку найкоротшого шляху у графі має важливе практичне значення. Наприклад, у GPS-навігаторах здійснюється пошук найкоротшого шляху між двома об'єктами. Перехрестя вулиць є аналогом вершин, а вулиці (дороги) з їхніми довжинами — ребрами. Якщо сума довжин доріг між об'єктами мінімальна, то вважається, що знайдено найкоротший шлях.

Роль ваги ребра можуть виконувати не тільки самі їхні довжини, а й час переміщення, витрачені фінансові, паливо-енергетичні ресурси, вартість та час виконання робіт тощо.

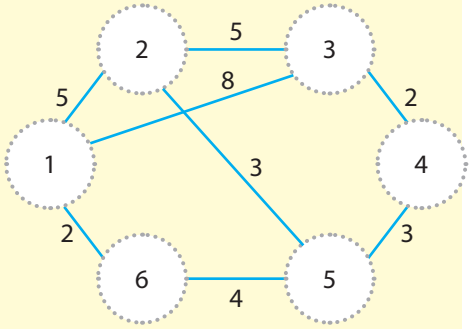


Рис. 8.19. Неорієнтований зважений граф

Якщо вага ребер не вказана, то найкоротшим шляхом вважається той, що містить найменшу кількість ребер. На графі, зображеному на рис. 8.19, найкоротшим шляхом із вершини 3 у вершину 6 є шлях 3–4–5–6, сума ваг ребер якого дорівнює 9.

Є різні варіанти задач про найкоротший шлях у графі:

- пошук найкоротшого шляху в задану вершину з усіх інших вершин графу;
- пошук найкоротшого шляху між двома заданими вершинами;
- пошук найкоротшого шляху з кожної вершини в усі інші вершини графу;
- пошук найкоротшого шляху, який проходить через задану множину вершин, та ін.

Нині для кожного типу задач пошуку найкоротшого шляху розроблено значну кількість алгоритмів. Розглянемо два з них: алгоритм Дейкстри й алгоритм Флойда — Уоршелла.

8.4.1. Алгоритм Дейкстри

Кур'єр доставляє кореспонденцію абонентам, користуючись схемою руху міського транспорту. Він майже щоденно відшукує найкоротший шлях від станції отримання кореспонденції до решти абонентів. Як він це робить?.

Алгоритм Дейкстри — алгоритм на графах, винайдений нідерландським ученим Едсгером Дейкстрою в 1959 році, знаходить найкоротший шлях від однієї вершини графу до всіх інших вершин.

За допомогою алгоритму Дейкстри здійснюється пошук найкоротшого шляху від однієї вершини графу (стартової) до всіх його інших вершин.

Для полегшення опису алгоритму Дейкстри будемо називати вершину *відвідуваною*, якщо вона вже була включена в будь-який шлях і її не можна більше включати в новий шлях, а вершину, яка ще не використовувалася на шляху і яка є суміжною з поточною, — *видимою*.

Наприклад, якщо вершина 0 (рис. 8.20) вже була включена в маршрут, то вона є відвідуваною, а якщо вершина 2 не включена в маршрут, то для неї видимими є вершини 3, 4 і 5.

Ви знаєте, що вершини графу можуть позначатися різними ідентифікаторами. Але якщо передбачається реалізувати алгоритм мовою Python, то доцільніше вершини позначати цілими числами в діапазоні від 0 до $n-1$. У цьому випадку можна використовувати номери вершин як індекси.

Сутність алгоритму Дейкстри така.

Крок 1	На кожному поточному кроці аналізуються всі видимі вершини і вибирається та, до якої відстань є найменшою. Ця вершина вважається поточною.
Крок 2	Далі перераховуються всі відстані до всіх видимих і відвідуваних вершин. У випадку якщо знайдеться менша відстань, попереднє значення замінюється меншим.

На прикладі графу, зображеного на рис. 8.20, знайдемо за допомогою алгоритму Дейкстри мінімальні відстані від вершини 0 до всіх інших вершин цього графу.

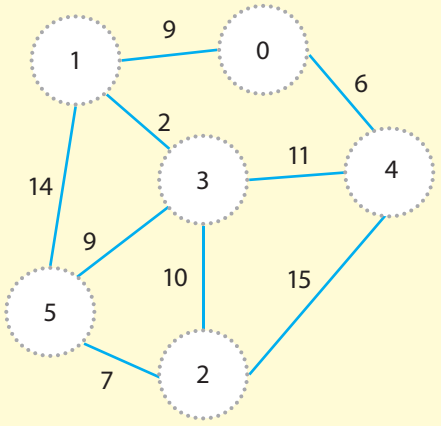


Рис. 8.20. Зважений неорієнтований граф

Приклад

1. Будемо вважати стартовою вершину 0, а кінцевою — вершину 5. Із вершини 0 видимими є вершини 1 і 4, відстані до яких відповідно дорівнюють 9 і 6. Інші вершини є невидимими, відстань до них невідома, що позначається великим числом, яке гарантовано буде більше чисел, які використовуються у процесі розв'язання цієї задачі (ми використали число 9999). Цей стан зафіксовано у табл. 8.1.

Таблиця 8.1. Відстані з вершини 0

Номер вершини графу	0	1	2	3	4	5
Поточна відстань від стартової вершини	0	9	9999	9999	6	9999
Вершина, з якої було зроблено останній перерахунок найменшої відстані	0	0	0	0	0	0

Отже, у результаті реалізації першого кроку відвідуваною є вершина 0 і визначено найменші відстані до видимих вершин.

2. Визначимо вершину, до якої можна перейти з вершини 0. Такими є вершини 1 і 4. Відстань до вершини 4 менша, тому переходимо у вершину 4. Із вершини 4 можна потрапити у вершини 2 і 3 (вершина 0 є відвідуваною, тому вона не аналізується), відстані до яких відповідно дорівнюють 15 і 11. Перераховуємо відстані до вершин 2 і 3 з вершини 4. Отримані дані відображено в табл. 8.2.

Таблиця 8.2. Відстані з вершини 4

Номер вершини графу	0	1	2	3	4	5
Відстань від стартової вершини	0	9	21	17	6	9999
Вершина, з якої було зроблено останній перерахунок	0	0	4	4	0	0

Таким чином, після цього кроку відвідуваними є вершини 0 і 4. Найближчою до вершини 4 є вершина 3, тому перейдемо у вершину 3.

3. Із вершини 3 видимими є вершини 1, 2, 5. Почнемо аналіз із вершини 1. Відстань до неї дорівнює $19\,(17+2)$. Але до цієї вершини зареєстрована відстань, що дорівнює 9. Тому відстань із вершини 1 до вершини 3 дорівнює 11. Цю відстань занесено в табл. 8.3.

Таблиця 8.3. Відстані з вершини 3

Номер вершини графу	0	1	2	3	4	5
Відстань від стартової вершини	0	9	21	11	6	9999
Вершина, з якої було зроблено останній перерахунок	0	0	4	1	0	0

Відстань до вершини 2 з вершини 3 дорівнює 27, тобто більше, ніж зареєстрована відстань до неї з вершини 4. Тому значення 21 не змінюємо. Таким чином, після цього кроку вибраними є вершини 0, 1, 2, 3, 4. Поточна довжина до них зареєстрована у табл. 8.3.

4. Із вершини 1 у вершину 3 відстань на поточний момент дорівнює 11. Із вершини 3 у вершину 5 відстань дорівнює 20, а з вершини 1 у цю ж вершину — 23. Тому обираємо перший варіант і записуємо у табл. 8.4.

Таблиця 8.4. Відстані від вершини 0 до всіх інших вершин

Номер вершини графу	0	1	2	3	4	5
Відстань від стартової вершини	0	9	21	11	6	20
Вершина, з якої було зроблено останній перерахунок	0	0	4	1	0	3

Відповідь: мінімальні відстані з вершини 0 у всі інші вершини (в порядку їх номерів) є такими: 9, 21, 11, 6, 20.

Алгоритм Дейкстри в загальному вигляді можна подати так.

Крок 1	Початок.	Крок 6	Надати поточній вершині статус відвідуваної.
Крок 2	Вибрати стартову вершину і визначити її поточною з номером *i*.	Крок 7	Перейти до п. 3.
Крок 3	Перевірити, чи всі вершини графу відвідані. Якщо всі, то перейти до п. 8, інакше — до п. 4.	Крок 8	Вивести шлях від фінішної вершини до стартової. Обчислити найкоротшу відстань між цими вершинами і вивести її.
Крок 4	Вибрати серед усіх видимих вершин ту, до якої найменша відстань, і визначити її поточною.		
Крок 5	Перерахувати відстані до всіх видимих і відвіданих вершин через вершину *i*. Якщо знайдеться менша відстань, замінити нею попереднє значення, а також присвоїти номер *i* цій вершині.	Крок 9	Кінець.

Для реалізації алгоритму Дейкстри мовою Python зважений неорієнтований граф можна реалізувати словником суміжності. У такому разі як ключ можна вибрати номер вершини, як значення — вагу ребра.

Опис графу (рис. 8.20) має такий вигляд:

```
N = [
    {1: 9, 4: 6},                  # Для нульової вершини
    {0: 9, 3: 2, 5: 14},           # Для першої вершини
    {3: 10, 4: 15, 5: 7},          # Для другої вершини
    {1: 2, 2: 10, 4: 11, 5 : 9},   # Для третьої вершині
    {0: 6, 2: 15, 3: 11},          # Для четвертої вершини
    {1: 14, 2: 7, 3: 9},           # Для п'ятої вершини
[
```

Отже, вершина 0 суміжна з вершиною 1, вага їхнього ребра дорівнює 9, а також суміжна з вершиною 4, вага ребра 6. Вершина 1 суміжна з нульовою, вага ребра дорівнює 9, з вершиною 3 з вагою ребра 2 і з п'ятою з вагою ребра 14 і т. д.

Розробимо програму реалізації алгоритму Дейкстри, у якій використано дані графу (рис. 8.20). Стартовою є вершина 0.

У програмі (рис. 8.21) використані такі змінні: p — словник поточної відстані до видимих вершин; u — список відвіданих вершин; min_x — номер вершини, до якої відстань є мінімальною від останньої відвіданої вершини; min_v — мінімальна відстань до поточної відвіданої вершини.

Уперше теорію графів як окрему математичну дисципліну представив угорський математик Кенінг у 1930-ті роки.

```
# Алгоритм Дейкстри. Пошук найкоротшого шляху з вершини 0 до всіх інших
def func (graph, start, p = {}, u =[]):    # Функція з параметрами
    if len(p) == 0: p[start] = 0           # Ініціалізація початкового шляху
    for x in graph[start]:                 # Цикл аналізу відстаней від вершини min_x
        if(x not in u and x != start):     # Якщо вершина не відвідана і не стартова
# Умова заміни попередньої відстані меншим значенням
            if (x not in p.keys() or (graph[start][x]+p[start])<p[x]):
                p[x] = graph[start][x]+p[start]      # Зміна значення відстані
    u.append(start)                        # Додавання вершини до списку відвідуваних
    min_v = 0                              # Початкове значення відстані
    for x in p:                            # Цикл аналізу відстаней до всіх вершин
        if (p[x]<min_v or min_v == 0) and x not in u:    # Умова вибору min значень
            min_x = x                      # Початкове значення номера вершини
            min_v = p[x]          # Мінімальна відстань до вершини, що аналізується
    if (len(u)<len(graph) and min_x):      # Якщо не відвід. усі верш. і не дорівнює 0
        return func (graph, min_x, p, u):  # Повернення до функції
    else:                          # Якщо усі вершини відвідані і поточна не є стартовою
        return p                   # Повернення результату
N =[                               # Ініціалізація графу за допомогою словника
    {1: 7, 2: 9 ,5: 14},           # Вершини з вагою ребер, суміжні з нульовою вершиною
    {0: 7, 2: 10, 3: 15},          # Вершини з вагою ребер, суміжні з першою вершиною
    {0: 9, 1: 10, 3: 11, 5: 2},    # Вершини з вагою ребер, суміжні з другою вершиною
    {1: 15, 2: 11, 4: 6},          # Вершини з вагою ребер, суміжні з третьою вершиною
    {3: 6, 5: 9}                   # Вершини з вагою ребер, суміжні з четвертою вершиною
    {0: 14, 2: 2, 4: 9}            # Вершини з вагою ребер, суміжні з п'ятою вершиною
]
print ('Мінімальна відстань до вершин. Перша цифра – вершина, друга – значення')
print (func(N,0))                  # Звернення до функції і виведення результату
```

Рис. 8.21. Програма реалізації алгоритму Дейкстри

Результат виконання програми:

```
Мінімальні відстані до вершин. Перша цифра – вершина, друга – значення
{0: 0, 1: 7, 2: 9, 3: 20, 4: 20, 5: 11}
```

8.4.2. Алгоритм Флойда — Уоршелла

Інкасатор щоденно відвідує мережу магазини і може потрапляти від будь-якого магазину до всіх інших найкоротшим шляхом. У такому випадку найкоротшу відстань можна знайти за допомогою алгоритму Флойда — Уоршелла.

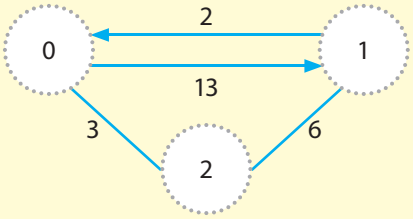


Рис. 8.22. Змішаний зважений граф

Ви знаєте, що для визначення найкоротшого шляху графу можна скористатися алгоритмом Дейкстри і в циклі обійти всі вершини. Розглянемо алгоритм Флойда — Уоршелла, який відрізняється від нього тим, що дозволяє знайти найкоротшу відстань не тільки від однієї вершини графу до всіх інших, а й від кожної вершини до всіх інших.

Згадаємо, що графи, які можуть одночасно використовуватися і дуги, і ребра, називають *змішаними*. Розглянемо сутність алгоритму Флойда — Уоршелла на прикладі змішаного зваженого графу (рис. 8.22). Сформулюємо алгоритм у загальному вигляді.

Приклад

1. Створимо для цього графу матрицю суміжності D[][].

Номер вершини	0	1	2
Вершина 0	0	13	3
Вершина 1	2	0	6
Вершина 2	3	6	0

У кожному елементі D[i][j] матриці зберігається вага ребра між вершинами *i* та *j*. У процесі реалізації алгоритму Флойда — Уоршелла будуть аналізуватися всі можливі варіанти довжин між вершинами *i* та *j*. Якщо виявлятиметься, що довжина менша, ніж зафіксована у поточний момент в елементі D[i][j], то менше значення замінить поточне.

Отже, у результаті повної реалізації алгоритму ми повинні отримати таку матрицю суміжності, у якій будуть зберігатися мінімальні довжини між усіма вершинами графу.

2. Зрозуміло, що довжина шляху з кожної вершини в ту саму вершину дорівнює 0 (навіть коли вершина має петлю). Наприклад, довжина шляху з вершини 2 у вершину 2 дорівнює 0. Це означає, що всі елементи D[i][i] дорівнюють нулю.

3. Якщо у графі є вершини, які між собою безпосередньо не мають ребер, то відповідним елементам матриці суміжності надамо нескінченне значення або максимально можливе, яке гарантовано буде більше від найдовшого шляху в графі. Наприклад, якщо вершини *i* та *j* безпосередньо не пов'язані ребром, то елементу D[i][j] надамо значення 9999. У графі, зображеному на рис. 8.22, таких вершин немає.

4. Найкоротший шлях із вершини *i* у вершину *j* може бути як безпосередньо між цими вершинами, так і через *k* інших вершин. Наприклад, довжина ребра безпосередньо між вершинами 1 і 2 дорівнює 6, а довжина ребер з вершини 1 у вершину 2 через вершину 0 — 5. У цьому випадку D[1][0]+D[0][2]<D[1][2]. Тому значення елемента D[1][2], яке на даний момент дорівнює 6, замінюється значенням довжини D[1][0]+D[0][2], яке дорівнює 5.

У загальному випадку реалізується таке правило:

якщо D[i][k]+D[k][j]<D[i][j], то значення елемента D[i][j] замінюється значенням D[i][k]+D[k][j].

5. Далі наведено детальний аналіз обчислення довжини шляху між усіма вершинами *i* та *j* через усі вершини *k*.

Вершини			Між якими вершинами обчислюється довжина шляху і через які вершини
к	і	j	
0	0	0	D[0][0] Не змінюється
0	0	1	D[0][1] Не змінюється
0	0	2	D[0][2] Не змінюється
0	1	0	D[1][0] Не змінюється
0	1	1	D[1][1] Не змінюється
0	1	2	3 1 → 2 через 0; D[1][2] змінюється на 5
0	2	0	D[2][0] Не змінюється
0	2	1	D[2][1] Не змінюється
0	2	2	D[2][2] Не змінюється
1	0	0	D[0][0] Не змінюється
1	0	1	D[0][1] Не змінюється
1	0	2	D[0][2] Не змінюється
1	1	0	D[1][0] Не змінюється
1	1	1	D[1][1] Не змінюється
1	1	2	D[1][2] Не змінюється
1	2	0	D[2][0] Не змінюється
1	2	1	D[2][1] Не змінюється
1	2	2	D[2][2] Не змінюється
2	0	0	D[0][0] Не змінюється
2	0	1	3 0 → 1 через 2; D[0][1] змінюється на 9
2	0	2	D[0][2] Не змінюється
2	1	0	D[1][0] Не змінюється
2	1	1	D[1][1] Не змінюється
2	1	2	D[1][2] Не змінюється
2	2	0	D[2][0] Не змінюється
2	2	1	D[2][1] Не змінюється
2	2	2	D[2][2] Не змінюється

6. Таким чином, після заміни значень елементів D[1][2] і D[0][1] матриця суміжності матиме такий зміст:

Номер вершини	0	1	2
Вершина 0	0	9	3
Вершина 1	2	0	5
Вершина 2	3	6	0

Алгоритм Флойда — Уоршелла — динамічний алгоритм для знаходження найкоротших шляхів між усіма вершинами зваженого орієнтованого графу.
Розроблений в 1962 году Робертом Флойдом і Стівеном Уоршеллом.
Уперше алгоритм розробив та опублікував Бернард Рой у 1959 році.

Розглянемо алгоритм Флойда — Уоршелла, поданий у загальному вигляді, для графу з кількістю вершин *n*.

Крок 1	Вибрати вершину k = 0, через яку здійснюється перерахунок довжини між вершинами і та j.
Крок 2	Вибрати вершину і = 0.
Крок 3	Вибрати вершину j = 0.

Крок 4	Якщо значення D[i][k]+D[k][j]<D[i][j], то D[i][j] = D[i][k]+D[k][j], інакше D[i][j] не змінюється.
Крок 5	Якщо j <= n, то j = j + 1 і перейти до п. 4, інакше — перейти до п. 6.
Крок 6	Якщо i <= n, то i = i + 1 і перейти до п. 3, інакше — перейти до п. 7.
Крок 7	Якщо k <= n, то k = k + 1 і перейти до п. 2, інакше — перейти до п. 8.
Крок 8	Кінець.

Програму реалізації алгоритму зображено на рис. 8.23.

```
# Програма реалізації алгоритму Флойда - Уоршелла
def func (D, n):                                    # Функція з двома параметрами
    for k in range (n):                             # Цикл вершини k
        for i in range (n):                         # Цикл вибору вершини i
            for j in range (n):                     # Цикл вибору вершини j
# Умова зміни поточної довжини між вершинами новим значенням
                if (D[i, k]!= 0) and (D[k, j]!= 0) and (i!= j):
                    if (D[i, k]+D[k, j]<D[i, j]) or (D[i, j] == 0):
                        D[i, j] = D[i, k]+D[k, j]    # Зміна поточної довжини
    for i in range (n):                             # Циклічне виведення
        for j in range (n):                         # Матриці найкоротших шляхів
            print (D[I, j], end =' ')               # Між вершинами графу
        print ()                                    # Перехід на наступний рядок
# Початок головного блоку програми
n = int(input ('Увести кількість вершин: '))        # Введення кількості вершин
graph = {}                                          # Порожній граф
print ('Уведення матриці суміжності')               # Повідомлення
# Початок циклічного уведення з клавіатури матриці суміжності
for i in range (n):                                 # Циклічне уведення з клавіатури
    for j in range (n):                             # ваги ребер
        print ('Увести GR[', i,',', j,']')          # Повідомлення про номер елемента
        graph[i, j] = int (input ())                # Уведення ваги ребер
print ('Матриця найкоротших шляхів:')               # Повідомлення
func (graph, n)                                     # Звернення до функції
```

Рис. 8.23. Програма реалізації алгоритму Флойда — Уоршелла

Запустимо програму і перевіримо її роботу на прикладі такої матриці суміжності:

Номер вершини	0	1	2	3
0	0	15	9999	2
1	4	0	10	3
2	9999	10	0	6
3	2	3	5	0

Динаміку та результат виконання програми наведено на рис. 8.24.

Запитання для перевірки знань

1. Назвіть основні варіанти задач пошуку найкоротшого шляху в графі.
2. Яка основна різниця між алгоритмами Дейкстри і Флойда — Уоршелла?
3. Поясніть порядок виконання алгоритму Дейкстри на прикладі найпростішого графу.
4. Поясніть порядок виконання алгоритму Флойда — Уоршелла на прикладі найпростішого графу.
5. Сформулюйте алгоритм Дейкстри.
6. Сформулюйте алгоритм Флойда — Уоршелла.

Завдання для самостійного виконання

1. Для графу на рис. 8.19 визначте найкоротші шляхи від вершини 2 до всіх інших.
2. Для графу на рис. 8.19 визначте найкоротші шляхи від кожної вершини до всіх інших.
3. Визначте найкоротший шлях з вершини A у вершину H у графі, зображеному на рис. 8.25.
4. Модифікуйте програму на рис. 8.21 для того, щоб вона визначала найкоротший шлях від вершини 0 до інших вершин для графу, зображеного на рис. 8.19. Виконайте програму і доведіть, що отримано правильні результати.
5. Модифікуйте програму на рис. 8.23 для того, щоб вона визначала найкоротший шлях від кожної вершини до всіх інших для графу, зображеного на рис. 8.19. Виконайте програму і доведіть, що отримано правильні результати.
6. Знайдіть в Інтернеті схему Київського метрополітену. Визначте мінімальну кількість станцій на шляху від станції «Вокзальна» до станції «Олімпійська».
7. Виконайте програму на рис. 8.23 для графу, зображеного на рис. 8.26.

```
Увести кількість вершин: 4
Уведення матриці суміжності
Увести GR[ 0, 0 ]
0
Увести GR[ 0, 1 ]
15
Увести GR[ 0, 2 ]
9999
Увести GR[ 0, 3 ]
2
Увести GR[ 1, 0 ]
4
Увести GR[ 1, 1 ]
0
Увести GR[ 1, 2 ]
10
Увести GR[ 1, 3 ]
3
Увести GR[ 2, 0 ]
9999
Увести GR[ 2, 1 ]
10
Увести GR[ 2, 2 ]
0
Увести GR[ 2, 3 ]
6
Увести GR[ 3, 0 ]
2
Увести GR[ 3, 1 ]
3
Увести GR[ 3, 2 ]
6
Увести GR[ 3, 3 ]
0
Матриця найкоротших шляхів:
0 5 8 2
4 0 9 3
8 9 0 6
2 3 6 0
```

Рис. 8.24. Динаміка та результат виконання програми

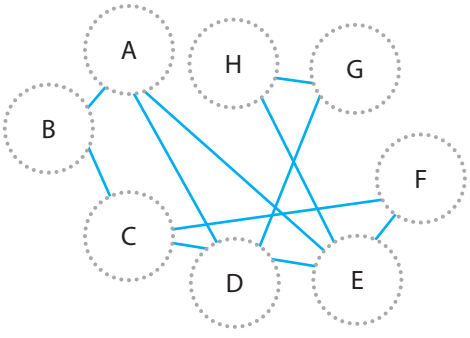


Рис. 8.25. Неорієнтований незважений граф

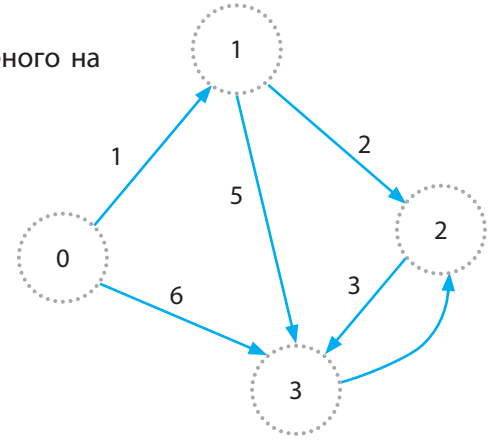


Рис. 8.26. Орієнтований зважений граф

9. Динамічне програмування і жадібні алгоритми

9.1. Динамічне програмування

Чи доводилося вам раніше у процесі вивчення математики або фізики розбивати складну задачу на декілька дрібніших однотипних задач?

Динамічне програмування — це метод розв'язування задач, що мають певні властивості, шляхом їх розбиття на декілька однотипних підзадач, пов'язаних між собою.

Основними типами задач динамічного програмування є задачі оптимізації, комбінаторики, задачі з використанням графів. Зазвичай задачі комбінаторики пов'язані з пошуком кількості об'єктів, які мають задані властивості, а задачі з використанням графів — із пошуком, наприклад, мінімальної відстані між пунктами A і B на карті доріг, поданій у вигляді графу.

Динамічне програмування — розділ математичного програмування, що вивчає багатокрокові процеси пошуку оптимального вирішення складних завдань.

Застосовується при складанні програм рішення таких задач оптимізації, для яких процес пошуку рішення можна представити у вигляді деякої послідовності кроків, тобто знаходять оптимальне рішення на кожному кроці процесу і таким чином зводять рішення однієї складної задачі до вирішення великого числа значно менш складних.

До *задач оптимізації* відносять задачі пошуку максимальних або мінімальних значень певних функцій. Наприклад, пошук максимального значення прибутку від інвестування у відповідну галузь за багаторічний період зводиться до послідовного визначення оптимальних капіталовкладень на кожен рік.

Загальним для задач динамічного програмування є те, що змінні в моделі розглядаються не разом, а послідовно, одна за одною, тобто будується така обчислювальна схема, коли замість одного завдання з багатьма змінними використовують багато завдань з малим числом (зазвичай навіть однієї) змінних у кожній. Це значно скорочує обсяг обчислень. Пояснимо сутність динамічного програмування на прикладі 1.

Поняття «динамічне програмування» було використано на початку 1940-х років американським математиком Річардом Беллманом. Він використав його для опису процесу пошуку розв'язку задач, для яких відповідь на одну задачу можна отримати лише після розв'язання попередньої задачі. У 1953 році вчений зробив уточнення терміна. Його стали використовувати для назви задач, безпосередньо пов'язаних із розв'язуванням вкладених підзадач для пошуку розв'язку всієї задачі. Нині термін розглядається як підрозділ системного аналізу та програмної інженерії.

Приклад 1. Нехай потрібно обчислити суму n чисел: $1+2+3+4+\ldots+n$. Складність цієї задачі полягає в тому, що необхідно одразу взяти велику кількість чисел і додати їх.

Можна спочатку визначити суму першого елемента, тобто просто взяти перший елемент: $f(1)=1$. Потім до суми можна додати другий елемент: $f(2)=f(1)+2=3$.

Аналогічно отримуємо: $f(3)=f(2)+3=6$, $f(n)=f(n-1)+n$.

Часто більшість виділених підзадач однакові. Але динамічне програмування дозволяє розв'язувати не всі типи задач, а лише певний клас задач, який використовується у таких галузях, як математика, програмування, статистика, економіка, лінгвістика, теорія ігор тощо, і які мають властивість *співоптимальності*.

Приклад 2. Припустимо, що компанія виготовляє процесори (позначимо їх змінною $x1$), системні плати ($x2$), монітори ($x3$) та інші пристрої. Для визначення максимального прибутку потрібно визначити, яку кількість пристроїв слід виготовити, скільки потрібно мати персоналу тощо.

Знайти розв'язок цієї задачі за великої кількості змінних досить складно. Але можна відокремити задачу із змінною $x1$. Після того як знайдено оптимальний розв'язок для першої підзадачі, виділяється підзадача з двома змінними $x1$ і $x2$, і вона розв'язується за допомогою вже знайденого розв'язку для першої задачі.

Потім виділяється підзадача з трьома змінними $x1$, $x2$ і $x3$ і розв'язується за допомогою раніше використаного методу.

Сутність співоптимальності полягає в тому, що велика складна задача поділяється на окремі невеликі підзадачі. Схематично цю сутність можна пояснити на прикладі логістики (розроблення оптимального плану виробництва).

Важливо те, що у процесі розв'язування складної задачі не розв'язується заново чергова підзадача, а використовується вже отриманий розв'язок попередньої підзадачі.

Наведемо типові задачі, які можна розв'язувати за допомогою динамічного програмування:

- розроблено граф доріг між населеними пунктами певного регіону; потрібно знайти мінімальний маршрут між двома населеними пунктами або маршрут, на який витрачається мінімальна кількість пального;
- корпорація виготовляє процесори, смартфони, монітори та інше обладнання; потрібно визначити оптимальну кількість виготовлення кожного виду пристроїв для отримання максимального прибутку.

В основу динамічного програмування покладено ідею розв'язання однієї складної задачі шляхом ділення її на окремі частини (підзадачі, етапи), розв'язування цих підзадач і об'єднання отриманих результатів в одне загальне розв'язання.

Розглянемо розв'язування задачі на обчислення чисел Фібоначчі на прикладі 3.

Приклад 3. Нехай є сходи, по яких можна ступати по кожній сходинці або через одну (рис. 9.1). Скількома способами можна потрапити на i сходинку?

На першу сходинку можна потрапити одним способом, на другу — двома, на третю — трьома способами.

Кількість способів потрапити на сходинку i дорівнює сумі способів потрапляння на сходинки $i-1$ і $i-2$, тобто обчислюється за допомогою формули $f(a_1)=f(a_{i-1})+f(a_{i-2})$, яка фактично є формулою обчислення чисел Фібоначчі. Нагадаємо, що числа Фібоначчі, починаючи з третього, отримують шляхом складання двох попередніх чисел, а перше і друге число дорівнюють одиниці:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,...

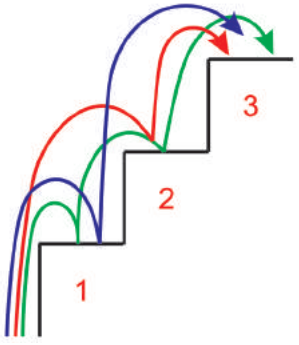


Рис. 9.1. Сходи

Щоб обчислити деяке число в цій послідовності, спочатку потрібно обчислити третє число, склавши перших два числа, потім четверте, склавши друге і третє числа, і т. д.

Здається, що задачу обчислення чисел Фібоначчі можна розв'язати методом перебору. Але цей метод не є ефективним. Проста рекурсивна функція вимагатиме суттєвих ресурсів вже на 30-му числі Фібоначчі. Під час застосування цього методу ті самі значення аргументів функції будуть обчислюватися багаторазово.

Програму мовою Python обчислення чисел Фібоначчі на основі рекурентного підходу наведено на рис. 9.2.

```
# Перший варіант програми обчислення чисел Фібоначчі та їх суми
def func (x):                              # Заголовок
    a, b = 1, 1                            # Початкові значення перших двох чисел
    while a<x:                             # Доти, доки a<x
        yield a                            # Повернення значення a
# Обчислюється b, потім a+b, значення яких присвоюються відповідно a і b
        (a, b) = (b, a+b)
s = 0                                      # Початкове значення суми чисел
print ('Числа Фібоначчі:')                 # Виведення повідомлення
for n in func (60):                        # Циклічне звернення до функції
    s = s+n                                # Поточне значення суми чисел
    print(n, end =" ")                     # Виведення чисел Фібоначчі
print ()                                   # Перехід на новий рядок
print ('Сума чисел:')                      # Виведення повідомлення
print ('s =',s)                            # Виведення повідомлення
```

Рис. 9.2. Перший варіант програми обчислення чисел Фібоначчі

Ще один варіант програми обчислення чисел Фібоначчі наведено на рис. 9.3.

```
# Другий варіант програми обчислення чисел Фібоначчі та їх суми
def func (n):                              # Функція з параметром
    a1 = a2 = 1                            # Значення перших двох чисел
    i = 2                                  # Початковий номер числа
    print ('Числа Фібоначчі:')             # Виведення повідомлення
    print (a1, a2)                         # Виведення значень перших двох чисел
    s = 2                                  # Початкове значення суми чисел
    while i<n:                             # Доти, доки i<n
        fib = a1+a2                        # Обчислення поточного значення числа
        a1 = a2                            # Зміна значення передостаннього числа
        a2 = fib                           # Зміна значення останнього числа
        s = s+fib                          # Обчислення поточного значення суми чисел
        i += 1                             # Поточний номер числа
        print (fib, end =" ")              # Виведення чисел, починаючи з третього
    print ()                               # Перехід на новий рядок
    print ('Сума чисел Фібоначчі:')        # Виведення повідомлення
    print ('s = ', s)                      # Виведення значення суми чисел
n = int(input ('Уведіть кількість чисел: '))     # Введення кількості чисел
func (n)                                         # Звернення до функції
```

Рис. 9.3. Другий варіант програми обчислення чисел Фібоначчі

Числа Фібоначчі можна обчислити в іншій спосіб: позбавитися обчислень, що повторюються. Це можна зробити за допомогою динамічного програмування — задача поділяється на підзадачі, що повторюються, кожна з яких розв'язується тільки один раз. Найпростіший варіант реалізації задачі — це запам'ятовувати обчисленні значення в додатковому масиві та використовувати їх для подальших обчислень.

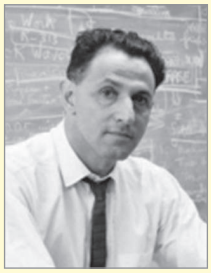
Розглянемо ще один приклад динамічного програмування — задачу про повернення решта.

Задачу повернення решти можна сформулювати так: визначити, скількома способами F(n) можна повернути решту суми n монетами заданого номіналу.

Є два варіанти цієї задачі.

1-й варіант: з урахуванням порядку видачі монет (купюр). Наприклад, решта у такому порядку видачі монет: 1, 2, 5 та 5, 2, 1 вважається різними варіантами.

Опишемо сутність застосування 1-го варіанта розв'язування на прикладі 4.



Річард Ернст Беллман — американський математик, один з найвидатніших фахівців в області математики та обчислювальної техніки. Зокрема його вважають батьком динамічного програмування.

Приклад 4. Нехай потрібно повернути решту 10 копійок монетами номіналом 1, 2 і 5 копійок. Позначимо F(10) кількість способів повернення здачі.

Спочатку розглянемо окремі випадки. Очевидно, що решту в одну копійку можна повернути одним способом, тобто повернути 1 копійку. Дві копійки можна повернути двома способами, а саме: 1+1 і 2 копійки. Три копійки можна повернути трьома способами: 1+2, 2+1 і 1+1+1. Решту в чотири копійки можна повернути п'ятьма способами: 2+2, 2+1+1, 1+2+1, 1+1+2 і 1+1+1+1.

Отже, ми визначили початкові значення: $F(1)=1$, $F(2)=2$, $F(3)=3$ і $F(4)=5$.

Спробуємо тепер узагальнити процес повернення решти. Кількість способів, якими можна повернути 10 копійок, залежить від того, яку монету повернуто першою. Якщо першою повернуто 1 копійку, то необхідно повернути ще 9 копійок; якщо першою повернуто 2 копійки, то слід повернути ще 8 копійок; якщо першою повернуто 5 копійок, то слід повернути ще 5 копійок. Кількість способів повернення 10 копійок дорівнює: $F(10)=F(9)+F(8)+F(5)$.

Отже, для визначення $F(10)$ необхідно знайти $F(9)$. Зрозуміло, що у випадку повернення першої монети номіналом 1 копійка необхідно повернути ще 8 копійок; якщо першою повертають 2 копійки, то необхідно повернути 7 копійок; якщо повернуто 5 копійок, то слід повернути ще 4 копійки.

Таким чином, кількість способів повернення решти в 9 копійок дорівнює:

$F(9)=F(8)+F(7)+F(4)$.

Звідси випливає, що повернення суми розміром k копійок монетами номіналом 1, 2 і 5 копійок обчислюється за формулою:

$F(k)=F(k-1)+F(k-2)+F(k-5)$ за умови, що $k\geqslant 5$ і $F(0)=1$, $F(1)=1$, $F(2)=2$, $F(3)=3$ і $F(4)=5$.

Продемонструємо порядок реалізації цієї формули:

$$F(5)=F(0)+F(3)+F(4)=F(0)+F(3)+F(4)=$$
$$=1+3+5=9.$$

$$F(6)=F(1)+F(4)+F(5)=1+5+9=15.$$

$$F(7)=F(2)+F(5)+F(6)=2+9+15=26.$$

$$F(8)=F(3)+F(6)+F(7)=3+15+26=44.$$

$$F(9)=F(4)+F(7)+F(8)=5+26+44=75.$$

$$F(10)=F(5)+F(8)+F(9)=9+44+75=128.$$

Програму реалізації цього алгоритму повернення решти подано на рис. 9.4.

```
# Кількість варіантів повернення решти з урахуванням порядку видачі монет
k = int(input ('Яку слід повернути здачу: '))       # Введення суми решти
F = [0]*(k+1)                                       # Створення порожнього масиву
F[0] = 1; F[1] = 1; F[2] = 2; F[3] = 3; F[4] = 5    # Початкові дані
for i in range (5, k+1):                            # Організація циклічного обчислення
    F[i] = F[i-1]+F[i-2]+F[i-5]                     # Обчислення кількості варіантів
print ("Кількість варіантів = ")                    # Виведення кількості варіантів
```

Рис. 9.4. Програма обчислення кількості варіантів повернення решти

Результат виконання програми:

```
Яку слід повернути здачу: 13
Кількість варіантів =  634
```

2-й варіант: без урахування порядку видачі монет (купюр). Головне полягає не в тому, в якому порядку клієнт отримує монети, а які саме монети отримує. Наприклад, видача решти монетами 1, 2, 5 і монетами 2, 5, 1 вважається одним варіантом.

Опишемо сутність застосування 2-го варіанта на прикладі 5.

Приклад 5. Нехай у касі є монети 1, 2 і 5 копійок, якими слід повернути решту 7 копійок. Можливі такі варіанти:

$1+1+1+1+1+1+1=7$

$1+1+1+1+1+2=7$

$1+1+1+2+2=7$

$1+2+2+2=7$

$1+1+5=7$

$2+5=7$

Усього шість варіантів.

Аналіз виконаних дій дозволяє зробити висновок, що процес пошуку кількості варіантів повернення решти є циклічним і складається із множини вкладених циклів. Кількість таких циклів визначається кількістю типів монет. Кількість виконання кожного вкладеного циклу залежить від суми здачі й номіналу самої монети.

Програму визначення кількості варіантів повернення здачі й кількості монет кожного з номіналів 1, 2, 5 і 10 зображено на рис. 9.5.

```
# Повернення решти монетами
sum = int(input ('Для якої суми слід повернути решту: '))
kilkist = 0                    # Початкове значення кількості варіантів
print ('Кількість монет для решти номіналом 1, 2, 5, 10 коп.:')      # Повідомлення
for k1 in range (sum//1+1):                                  # Цикл для однієї копійки
    for k2 in range (sum//2+1):                              # Цикл для двох копійок
        for k5 in range (sum//5+1):                          # Цикл для п'яти копійок
            for k10 in range (sum//10+1):                    # Цикл для десяти копійок
                if k1+2*k2+5*k5+10*k10 == sum:               # Якщо можна повернути решту
                    a = [k1, k2, k5, k10]                    # Список варіанта решти
                    print (a)                                # Виведення варіанта решти
                    kilkist += 1                             # Збільшення кількості варіантів
print ('Усього варіантів задачі = ', kilkist)                # Виведення кількості варіантів
```

Рис. 9.5. Програма обчислення повернення решти

Результат виконання програми:

```
Для якої суми слід повернути решту: 12
Кількість монет для решти номіналом 1, 2, 5, 10 коп.:
[0, 1, 0, 1]
[0, 1, 2, 0]
[0, 6, 0, 0]
[1, 3, 1, 0]
[2, 0, 0, 1]
[2, 0, 2, 0]
[2, 5, 0, 0]
[3, 2, 1, 0]
[4, 4, 0, 0]
[5, 1, 1, 0]
[6, 3, 0, 0]
[7, 0, 1, 0]
[8, 2, 0, 0]
[10, 1, 0, 0]
[12, 0, 0, 0]
Усього варіантів решти =  15
```

Принцип оптимальності динамічного програмування сформулював Р. Беллман: «Оптимальна поведінка має таку властивість: якими б не були первісний стан і рішення в початковий момент, наступні рішення повинні бути оптимальними щодо стану, отриманого в результаті первісного рішення».
Внесок Беллмана в динамічне програмування був увічнений у назві «рівняння Беллмана», центрального результату теорії динамічного програмування.

Запитання для перевірки знань

1. Які задачі можна розв'язувати за допомогою динамічного програмування?
2. У чому полягає сутність властивості співоптимальності?
3. Які типи задач можна розв'язувати за допомогою динамічного програмування?
4. Поясніть на прикладі сутність динамічного програмування.

Завдання для самостійного виконання

1. Визначте, скількома способами можна сплатити суму 60 грн купюрами номіналом 5, 10, 20 і 50 грн.
2. Виконайте програму, зображену на рис. 9.3, для отримання суми перших 15 чисел Фібоначчі. Доведіть, що отримано правильний результат.
3. Модифікуйте програму, наведену на рис. 9.2, таким чином, щоб обчислювалася тільки сума перших 12 чисел Фібоначчі.
4. Модифікуйте програму, наведену на рис. 9.4, таким чином, щоб вона підраховувала всі варіанти повернення здачі 17 копійок монетами 2, 5 і 10 копійок і кількість варіантів.
5. У корпорації три компанії. На початку року між цими компаніями розподіляють-ся 200 млн грн інвестицій у сумі, кратній 50 млн.

   У таблиці показано приріст продукції (у млн грн) залежно від розміру інвестицій.

Сума інвестицій	Приріст продукції від компаній		
	1-ша компанія	2-га компанія	3-тя компанія
0	9	14	12
50	12	19	16
100	15	21	20
150	20	24	25
200	24	29	27

   Розподіліть між компаніями 200 млн грн так, щоб отримати максимальний загальний прибуток.
6. Поле розбито на клітинки з **n** стовпцями і **m** рядками. По полю можна крокувати на одну клітинку вправо і вгору. Розробіть програму обчислення кількості способів потрапляння з нижнього лівого кута в правий верхній.

9.2. Жадібні алгоритми

Пригадайте випадки, коли вам доводилося з множини можливих варіантів вибирати той, який є найкращим у даний момент, не хвилюючись про те, що цей варіант може не призвести до найкращого результату після завершення всього процесу.

Жадібні алгоритми — це загальна назва методів розв'язування задач оптимізації, при яких на кожному етапі приймається локально оптимальне рішення.

Незважаючи на те що такий підхід не завжди забезпечує оптимальне розв'язання задачі в цілому, в багатьох випадках досягаються результати, близькі до оптимальних.

Жадібний алгоритм — це метод розв'язування оптимізаційних задач, заснований на тому, що процес прийняття рішення можна розбити на елементарні кроки, на кожному з яких приймається окреме рішення.

Ці алгоритми застосовують до тих задач, які можна розбити на окремі прості підзадачі аналогічно тому, як це робиться в алгоритмах динамічного програмування.

У жадібних алгоритмах на кожному етапі з множини можливих варіантів вибирається той, який є найкращим у даний момент, із надією, що саме він забезпечить оптимальне розв'язання всієї задачі. Вибір варіанта на кожному етапі повинен відповідати таким вимогам: бути допустимим (відповідати обмеженням задачі); бути оптимальним (найкращим із усіх можливих на даному етапі); бути остаточним (не можна змінити цей вибір на наступних етапах).

Пояснимо сутність жадібних алгоритмів на прикладі 1.

Приклад 1. Банкомат може видавати купюри номіналом 500, 200, 100, 50 і 20 грн. Якими купюрами і в якій кількості повинен видати банкомат суму 2340 грн, щоб кількість купюр була мінімальною?

Відповідно до методу жадібних алгоритмів програма обслуговування банкомата на кожному кроці повинна вибирати купюри максимального номіналу з числа тих, які можна вибрати, щоб їх загальна кількість була мінімальною. Для цієї суми спочатку слід вибрати чотири купюри номіналом 500 грн, потім одну купюру номіналом 200 грн, далі одну купюру номіналом 100 грн і, нарешті, дві купюри номіналом 20 грн. Усього 8 купюр.

Наведемо тепер приклад 2, який підтверджує, що жадібні алгоритми не завжди забезпечують оптимальне розв'язання задачі в цілому.

Приклад 2. Борошно міститься в контейнерах масою 110, 50 і 10 кг. Виберемо мінімальну кількість контейнерів загальною масою 150 кг. Логічно звяти 3 контейнери масою 50 кг.

За жадібним алгоритмом спочатку потрібно взяти контейнер максимальної маси (110 кг), на наступних етапах — 4 контейнери масою 10 кг, тобто 5 контейнерів.

Розглянемо приклад 3 про пакування рюкзака. Існують різні інтерпретації цієї задачі, але її загальна сутність одна.

Є декілька типів об'єктів і рюкзак, у який можна покласти певну кількість об'єктів одного або різних типів. Відома маса і ціна кожного типу об'єктів. Необхідно покласти в рюкзак такі типи об'єктів певної кількості, щоб їх маса не перевищувала допустиму масу рюкзака, а ціна об'єктів була максимальною.

Ідея жадібного алгоритму полягає в тому, щоб класти до рюкзака найефективніші предмети, аж до його заповнення. Для розв'язання цієї задачі можна застосувати й засоби динамічного програмування.

Приклад 3. Одним із найпростіших алгоритмів для розв'язування цієї задачі є жадібний алгоритм. Використаємо ідентифікатори для таких початкових даних:

n — кількість об'єктів, які можна покласти в рюкзак;

gruz — допустима маса рюкзака;

nazva — список типів об'єктів, які можна покласти в рюкзак;

zina — список цін об'єктів;

vaga — список ваги об'єктів.

Будемо вважати, що в наявності є об'єкти, з яких можна покласти в рюкзак рівно таку вагу, на яку його розраховано.

Крім перелічених, використаємо в алгоритмі такі ідентифікатори:

zina_1 — список цін одиниць об'єктів (наприклад, одного кг);

kilkist — список кількостей об'єктів кожного типу, покладених у рюкзак;

Masa — поточне значення маси об'єктів у рюкзаку;

Sum — ціна об'єктів, покладених у рюкзак.

Існують різні підходи до розроблення алгоритмів упакування рюкзака.

Будемо дотримуватися такого підходу, як наведено в алгоритмі.

Крок 1	Увести значення змінних n, gruz і списків nazva, zina, vaga і сформувати порожній список kilkist довжиною n.
Крок 2	Обчислити значення списку zina_1 за допомогою виразу zina_1[i] = zina[i]/vaga[i].
Крок 3	Упорядкувати список zina_1 у порядку зменшення значень його елементів і відповідно до цього упорядкувати елементи списків nazva, zina, vaga.
Крок 4	i = 0; Masa = 0; Sum = 0.
Крок 5	Доти, доки Masa<gruz, виконувати п. 6, інакше — п. 7.
Крок 6	kilkist[i] = kilkist[i]+1; Masa = Masa+vaga[i]; Sum = Sum+zina[i].
Крок 7	Якщо Masa == gruz, то п. 11, інакше — п. 8.
Крок 8	Відновити значення змінної Sum і списків kilkist, Masa і Sum.
Крок 9	i = i + 1.
Крок 10	Якщо i <= n, то п. 5, інакше — п. 11.
Крок 11	Вивести значення змінної Sum і списків nazva і kilkist.

Програму реалізації алгоритму пакування рюкзака зображено на рис. 9.6.

```
# Задача про рюкзак
gruz = int(input ('Вага рюкзака: '))                    # Допустима вага рюкзака
n = int(input ('Кількість назв об'єктів:'))             # Уведення кількості назв об'єктів
nazva = []                                              # Порожній список nazva
zina = []                                               # Порожній список zina
vaga = []                                               # Порожній список vaga
kilkist = []                                            # Порожній список kilkist
zina_1 = []                                             # Порожній список zina_1
for i in range (n):                                     # Цикл уведення значень списку
    kilkist.append (0)                                  # Заповнення нулями списку
    b = str(input ('Назва об'єкта: '))                  # Уведення назви об'єкта
    nazva.append (b)                                    # Додавання до списку назви об'єкта
    b = int(input ('Ціна: '))                           # Уведення ціни об'єкта
    zina.append (b)                                     # Додавання до списку ціни об'єкта
    b = int(input ('Вага: '))                           # Уведення ваги об'єкта
    vaga.append (b)                                     # Додавання до списку ваги об'єкта
    zina_1.append (zina[i]/vaga[i])                     # Ціна одиниці об'єкта
p = n -1                                                # Початкове значення правої межі
while p>0:                                              # Зовнішній цикл сортування масиву
    y = True                                            # Ознака відсутності переміщень
    for i in range (p):                                 # Внутрішній цикл сорт. масиву
        if zina_1[i]<zina[i+1]:                         # Попередній менше наступного?
            z = zina_1[i]; zina_1[i] = zina_1[i+1]; zina_1[i+1] = z    # Переміщення
            z = zina[i]; zina[i] = zina[i+1]; zina[i+1] = z            # значень
            z = nazva[i]; nazva[i] = nazva[i+1]; nazva[i+1] = z        # елементів
            z = vaga[i]; vaga[i] = vaga[i+1]; vaga[i+1] = z            # списків
            y = False                                   # Ознака наявності переміщень
    p = p-1                                             # Зменшення правої межі
Masa = 0                                                # Початкова вага рюкзака
Sum = 0                                                 # Початкова ціна рюкзака
for i in range (n):                                     # Зовнішній цикл
    while (Masa>gruz):                                  # Внутрішній цикл
        kilkist[i] = kilkist[i]+1                       # Збільшення кількості і-го об'єкта
        Masa = Masa+vaga[i]                             # Збільшення ваги рюкзака
        Sum = Sum+zina[i]                               # Збільшення ціни рюкзака
    if Masa == gruz: break                              # Вага рюкзака дорівнює допустимій?
    Masa = Masa-vaga[i]                                 # Повернення попередньої ваги
    kilkist[i] = kilkist[i]-1                           # Повернення попередньої кількості
    Sum = Sum-zina[i]                                   # Повернення попередньої ціни
for i in range (n):                                     # Цикл виведення
    print (nazva[i], ' = ', kilkist[i], ' упаковок ')       # Назви і кількість
print ('Ціна рюкзака = ', Sum)                                 # Виведення ціни рюкзака
```

Рис. 9.6. Програма реалізації алгоритму про пакування рюкзака

На рис. 9.7 наведено варіант введення даних і отриманий результат.

Результат виконання програми:

```
Вага рюкзака: 15
Кількість назв об'єктів: 5
Назва об'єкта: шпроти
Ціна: 340
Вага: 2
Назва об'єкта: бички
Ціна: 80
Вага: 1
Назва об'єкта: форель
Ціна: 1200
Вага: 5
Назва об'єкта: тріска
Ціна: 240
Вага: 2
Назва об'єкта: осетер
Ціна: 1100
Вага: 4
осетер  =  3  упаковки
форель  =  0  упаковок
шпроти  =  1  упаковка
тріска  =  0  упаковок
бички  =  0  упаковок
Ціна рюкзака  =  3720
```

Рис. 9.7. Варіант введення даних і отриманий результат

Жадібні алгоритми можуть бути досить ефективними для простих задач. Для багатьох інших задач вони не лише не видають оптимального розв'язку, а й можуть видати один із найгірших варіантів. Жадібні алгоритми ефективні у випадку, якщо послідовність локально оптимальних виборів забезпечує глобальне оптимальне рішення.

Запитання для перевірки знань

1. До яких задач можна застосовувати жадібні алгоритми?
2. Чи завжди жадібні алгоритми дозволяють отримати оптимальне розв'язання?
3. У чому полягає сутність жадібних алгоритмів?
4. Наведіть приклад задачі, для якої жадібні алгоритми не забезпечують оптимальне розв'язання.
5. Поясніть сутність жадібних алгоритмів на конкретному прикладі.

Завдання для самостійного виконання

1. Змініть значення деяких початкових даних, які вводилися у процесі виконання програми, зображеної на рис. 9.6, і доведіть, що отримано правильний результат.
2. Визначте, як сплатити суму 91 грн купюрами номіналом 1, 2, 5, 10, 25 і 50 грн так, щоб загальна кількість купюр була мінімальною.
3. Сформулюйте самостійно таку умову задачі, щоб її можна було реалізувати програмою, зображеною на рис. 9.7. Виконайте цю програму для власних даних і доведіть, що отримано правильні результати.
4. Працівнику запропонували за відведений час виконати декілька видів робіт. Які роботи слід виконати працівникові, щоб отримати максимальний прибуток, якщо відомі час їх виконання і вартість цих робіт?
5. У палаці «Україна» в Києві протягом трьох годин триває виступ творчих колективів. Заявка надійшли від n колективів з указаним часом виступу. Розробіть програму визначення максимальної кількості колективів для участі у заході, та їхні назви.

10. Основи обчислювальної геометрії

10.1. Базові поняття

Пригадайте, які геометричні фігури ви використовували найчастіше під час вивчення геометрії.

У таких сферах, як робототехніка, системи автоматизованого програмування, комп'ютерна графіка тощо, застосовуються алгоритми, що описуються в термінах геометрії.

Обчислювальна геометрія — це галузь комп'ютерних наук, присвячена вивченню алгоритмів розв'язування геометричних задач.

У процесі розв'язування задач обчислювальної геометрії використовуються базові геометричні об'єкти: точка, відрізок, пряма, вектор, многокутник, коло.

Приклади геометричних задач: пошук координат точки перетину двох прямих; перевірка приналежності точки відрізку; перевірка приналежності точки многокутнику; обчислення площі многокутника; побудова опуклої оболонки.

Точка на площині в декартовій системі координат задається двома числами — координатами x і y, а у тривимірному просторі — трьома координатами (x', y', z'). Відрізок можна задати, вказавши координати його початку й кінця, пряму — вказавши координати двох її точок.

Оскільки в обчислювальній геометрії вектор є одним з основних «інструментів», згадаємо основні відомості про нього, що вивчалися на уроках геометрії.

Вектор — це напрямлений відрізок AB, де A — точка початку (її ще називають *точкою прикладання*), B — точка кінця відрізку.

Вектор позначають двома великими латинськими буквами зі стрілкою $(\overrightarrow{AB})$ або рискою зверху $(\overline{AB})$. Вектор також позначають однією буквою (a), або буквою зі стрілкою $(\vec{a})$, або буквою з рискою зверху $(\bar{a})$. Приклади зображення векторів та їх позначення наведено на рис. 10.1.

Вектор характеризується довжиною і напрямком.

Кожний вектор має проекцію на відповідну вісь. Проекцією вектора на вісь називають відрізок, кінцями якого є проекції точок початку й кінця вектора на задану вісь. Довжина проекції — довжина цього відрізка. Довжину проекції інколи також називають проекцією. Проекція має знак «плюс», якщо напрямок від проекції початку до проекції кінця вектора збігається з напрямком осі, і знак «мінус», якщо ці напрямки протилежні.

Проекція дорівнює довжині початкового вектора, помноженій на косинус кута між цим вектором і віссю (рис. 10.2).

Нехай точка A в декартовій системі координат має координати (x_1, y_1), а точка B — координати (x_2, y_2).

Термін «вектор» увів ірландський математик Вільям Ровен Гамільтон у середині XIX ст. Він також описав деякі операції векторного аналізу.

Теорію векторів використав шотландський учений Джеймс Максвелл у працях з електромагнетизму. У 1903 році англійський учений Олівер Гевісайд надав векторному аналізу сучасний вигляд.

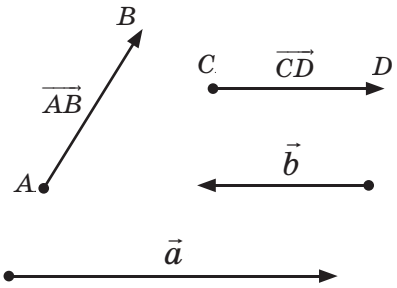


Рис. 10.1. Зображення векторів та їх позначення

Координатами вектора $(\overrightarrow{AB})$ називають пару чисел $(x_2 - x_1, y_2 - y_1)$. Якщо вектор має координати (x, y) і прикладений до точки (x_1, y_1), то можна обчислити координати його кінця (x_2, y_2) так: $x_2 = x_1 + x$, $y_2 = y_1 + y$.

Довжиною, або модулем, вектора називають відстань між точками його кінця і початку.

Довжина обчислюється за формулою $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

Для позначення довжини вектора часто використовують символи модуля: $|a|$.

Вектори бувають колінеарні й неколінеарні.

Колінеарні вектори — це вектори, що лежать на одній прямій або паралельних прямих (рис. 10.3). Колінеарні вектори, у свою чергу, поділяють на дві групи: однаково напрямлені (співнапрямлені) і протилежно напрямлені.

На рис. 10.3 вектори $\vec{a}$ і $\vec{b}$ однаково напрямлені, це позначається так: $\vec{a} \uparrow\uparrow \vec{b}$, а вектори $\vec{a}$ і $\vec{d}$ протилежно напрямлені, це позначаються так: $\vec{a} \uparrow\downarrow \vec{d}$.

Два вектори називають рівними, якщо вони співнапрямлені і мають однакову довжину. Два колінеарні вектори, що мають однакові довжини, але протилежні напрямки, називають протилежними. Вектор, початок і кінець якого збігаються, називають нульовим і позначають $\vec{0}$. Одиничним вектором, або ортом, називають вектор, довжина якого дорівнює одиниці.

Орт позначають буквою $\vec{e}$, його модуль $|\vec{e}| = 1$.

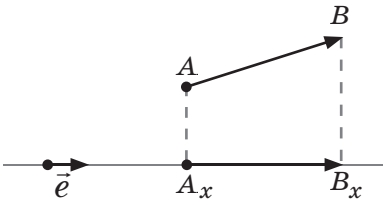


Рис. 10.2. Проекція вектора на вісь X

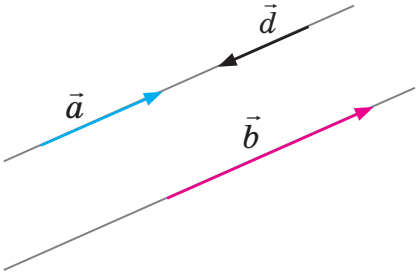


Рис. 10.3. Колінеарні вектори

Запитання для перевірки знань

1. Наведіть приклади задач обчислювальної геометрії.
2. Які базові об'єкти використовуються в обчислювальній геометрії?
3. Як можна задати пряму на площині?
4. Що таке вектор? Як його позначають?
5. Які вектори називають колінеарними; рівними?
6. Що називають проекцією вектора на вісь?
7. Як визначити координати вектора?
8. Як визначити модуль вектора?

Завдання для самостійного виконання

1. Точка A має координати $(2; 4)$, а точка B — координати $(5; 10)$. Наведіть приклади таких координат точок C і D, щоб вектор $\overrightarrow{CD}$ був колінеарним вектору $\overrightarrow{AB}$.
2. Точка A має координати $(-1; 4)$, а точка B — координати $(5; 9)$. Наведіть приклади таких координат точок C і D, щоб вектор $\overrightarrow{CD}$ був рівним вектору $\overrightarrow{AB}$.
3. Дано вектор $\vec{c}$, $|\vec{c}| = 10$, кут між вектором і віссю становить 60°. Визначте проекцію вектора $\vec{c}$ на цю вісь.
4. Дано вектор $\vec{a}$ з координатами початку $(3; 4)$ і кінця $(8; 6)$ і вектор $\vec{b}$ з координатами початку $(1; 7)$ і кінця $(5; 9)$. Визначте, чи є ці вектори колінеарними.
5. Точка A має координати $(-2; 4)$, точка B — координати $(1; 6)$. Визначте довжину вектора $\overrightarrow{BA}$.
6. Вектор $\vec{a}$ має координати $(3; 7)$ і прикладений до точки $(1; 4)$. Обчисліть координати точки кінця вектора.

10.2. Операції над векторами

З якою метою ви використовували вектори під час вивчення фізики? Чи доводилося вам використовувати вектори у процесі вивчення інших навчальних дисциплін?

Якщо вектори безпосередньо зображені на площині, то додавання двох векторів можна виконати за допомогою правил трикутника і паралелограма.

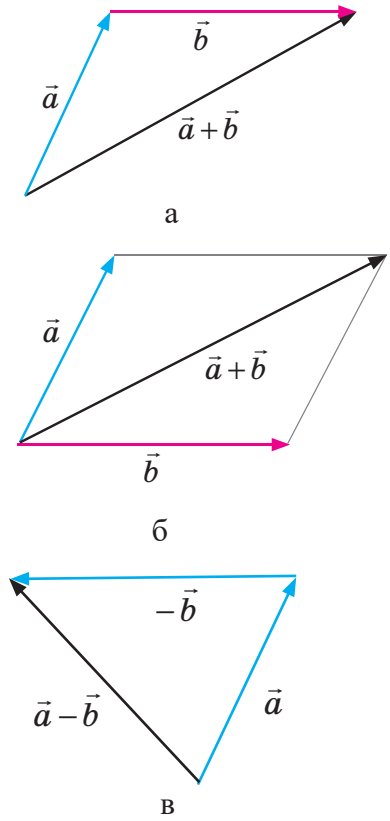


Рис. 10.4. Додавання і віднімання векторів

Над векторами можна виконувати операції додавання, віднімання, множення на число та ін.

Додавання та віднімання векторів. Якщо вектори задані координатами на площині: $\vec{a}=(a_x;a_y)$ і $\vec{b}=(b_x;b_y)$, то їхня сума обчислюється за формулою $\vec{a}+\vec{b}=(a_x+b_x;a_y+b_y)$, а різниця — за формулою $\vec{a}-\vec{b}=(a_x-b_x;a_y-b_y)$.

Для додавання векторів $\vec{a}$ і $\vec{b}$ за *правилом трикутника* слід перемістити вектор $\vec{b}$ паралельно самому собі так, щоб його початок збігся з кінцем вектора $\vec{a}$, тоді їхньою сумою буде вектор, початок якого збігається з початком вектора $\vec{a}$, а кінець — з кінцем вектора $\vec{b}$ (рис. 10.4, *а*).

Для додавання вектора за *правилом паралелограма* один з векторів переносять паралельно собі так, щоб початки векторів збіглися. Потім на їх основі будують паралелограм і проводять його діагональ (рис. 10.4, *б*).

Різницею векторів $\vec{a}$ і $\vec{b}$ є сума вектора $\vec{a}$ з вектором, протилежним вектору $\vec{b}$. Різницю векторів (рис. 10.4, *в*) можна знайти також за правилом трикутника або паралелограма. У цьому випадку напрям вектора $\vec{b}$ змінюють на протилежний, потім отриманий вектор переносять паралельно самому собі в кінець вектора $\vec{a}$. Початок вектора $\vec{a}$ з'єднують із кінцем вектора $\vec{b}$.

Приклад 1. Дано вектори $\vec{a}$ і $\vec{b}$ і кут між ними $90°$, модулі векторів $|\vec{a}|=3$ і $|\vec{b}|=4$. Тоді $|\vec{c}|=|\vec{a}+\vec{b}|=\sqrt{9+16}=5$.

Множення вектора на число. У результаті множення вектора $\vec{a}$ на число k кожна координата вектора множиться на число k. У результаті отримаємо вектор завдовжки $|k|\cdot|\vec{a}|$. Його напрям збігається з напрямом $\vec{a}$, якщо $k>0$, і має протилежний напрям, якщо $k<0$ (приклад 2).

Приклад 2. Дано вектор $\vec{a}=(5;-3)$. Тоді вектор $2\vec{a}=2(5;-3)=(2\cdot5;2\cdot(-3))=$ $=(10;-6)$.

Скалярний добуток. Скалярним добутком векторів $\vec{a}$ і $\vec{b}$ є число, що дорівнює добутку їх довжин на косинус кута між ними. Тобто скалярний добуток визначається за формулою $\vec{a}\cdot\vec{b}=|\vec{a}|\cdot|\vec{b}|\cos\alpha$, де α — кут між даними векторами (приклад 3).

Векторний добуток. Векторним добутком вектора $\vec{a}$ на вектор $\vec{b}$ називається вектор $\vec{c}$, $|\vec{c}|=|\vec{a}|\cdot|\vec{b}|\cdot\sin\alpha$, де α — кут між векторами $\vec{a}$ і $\vec{b}$. Отже, модуль векторного добутку векторів $\vec{a}$ і $\vec{b}$ чисельно дорівнює площі паралелограма (рис. 10.5), побудованого на цих векторах.

Векторний добуток найчастіше позначається так: $\vec{c}=\vec{a}\times\vec{b}$, або у квадратних дужках: $\vec{c}=[\vec{a},\vec{b}]$, $\vec{c}=[\vec{a}\times\vec{b}]$.

Під час виконання добутку вектора $\vec{b}$ на вектор $\vec{a}$ знак вектора $\vec{c}$ змінюється на протилежний.

Вектор $\vec{c}$ перпендикулярний (ортогональний) до площини, у якій лежать вектори $\vec{a}$ і $\vec{b}$, і напрямлений так, що поворот від вектора $\vec{a}$ до вектора $\vec{b}$ здійснюється проти ходу годинникової стрілки, якщо дивитися з кінця вектора $\vec{c}$. У такому випадку говорять, що трійка векторів $\vec{a}$, $\vec{b}$ і $\vec{c}$ — права.

Для визначення напрямку вектора $\vec{c}$ на практиці користуються правилом правої руки. Для цього вказівний палець слід направити у напрямку вектора $\vec{a}$, а середній — у напрямку вектора $\vec{b}$ (рис. 10.6). Якщо великий палець направлений вгору, то це означає, що вектор напрямлений від нас, а якщо вниз, то до нас (приклад 4).

Приклад 4. Нехай довжина вектора $\vec{a}$ дорівнює 5, довжина вектора $\vec{b}$ — 10, а кут між ними становить 30° Тоді довжина вектора $|\vec{c}|=5\cdot10\cdot\sin(30^\circ)=25$. Його напрямок можна визначити за правилом правої руки.

Векторний добуток довільного вектора на нульовий вектор дорівнює нульовому вектору. Векторний добуток двох колінеарних векторів також дорівнює нульовому вектору.

Приклад 3. Дано: $|\vec{a}|=4$, $|\vec{b}|=5$, $a=30^\circ$, тоді $\vec{a}\cdot\vec{b}=|\vec{a}|\cdot|\vec{b}|\cdot\cos30^\circ=10\sqrt{3}$. Якщо два вектори задані своїми координатами $(x_1;y_1)$ і $(x_2;y_2)$, то $\vec{a}\cdot\vec{b}=x_1x_2+y_1y_2$. Наприклад, якщо $\vec{a}=(2;5)$ і $\vec{b}=(-3;4)$, то $\vec{a}\cdot\vec{b}=2\cdot(-3)+5\cdot4=14$.

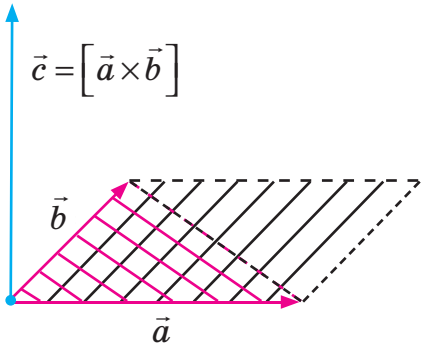


Рис. 10.5. Графічне зображення векторного добутку

Розглянемо тепер знаходження векторного добутку в координатах.

Нехай дано два вектори: $\vec{a}(a_1,a_2,a_3)$ і $\vec{b}(b_1,b_2,b_3)$. Координати векторного добутку $\vec{c}$ на практиці часто обчислюють за формулою: $\vec{c}(a_2\cdot b_3-a_3\cdot b_2;\ a_3\cdot b_1-a_1\cdot b_3;\ a_1\cdot b_2-a_2\cdot b_1)$ (приклад 5).

Приклад 5. Відомі координати двох векторів: $\vec{a}(-1;2;-3)$ і $\vec{b}(0;-4;1)$. Тоді координати векторного добутку є такими: $\vec{c}(2\cdot1-(-3)\cdot(-4);(-3)\cdot0-(-1)\cdot1;(-1)\cdot(-4)-2\cdot0)=\vec{c}(-10;1;4)$.

Якщо відомі координати вектора $\vec{c}(c_1,c_2,c_3)$, то його довжину можна визначити за формулою $|\vec{c}|=\sqrt{c_1^2+c_2^2+c_3^2}$. Довжина вектора $\vec{c}$ зі знайденими вище координатами дорівнює: $|\vec{c}|=\sqrt{(-10)^2+1^2+4^2}=3\sqrt{13}$.

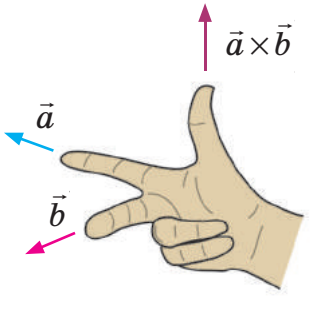


Рис. 10.6. Правило правої руки

На рис. 10.7 зображено програму обчислення координат векторного добутку двох векторів та його довжини.

```
# Визначення координат і довжини векторного добутку
import math                                  # Імпортування модуля math
mas_a = []                                   # Порожній список для координат першого вектора
mas_b = []                                   # Порожній список для координат другого вектора
for i in range (3):                          # Цикл уведення координат векторів
    print ('координата a(',i,') ')          # Повідомлення про коорд. першого вектора
    a = float (input())                      # Уведення координат першого вектора
    mas_a.append (a)                         # Список координат першого вектора
    print ('координата b(',i,') ')          # Повідомлення про коорд. другого вектора
    a = float (input())                      # Уведення координат другого вектора
    mas_b.append (a)                         # Список координат другого вектора
i = 1                                        # Початкове значення номера координати
c1 = mas_a[i]*mas_b[i+1]-mas_a[i+1]*mas_b[i]          # Перша координата
c2 = mas_a[i+1]*mas_b[i-1]-mas_a[i-1]*mas_b[i+1]      # Друга координата
c3 = mas_a[i-1]*mas_b[i]-mas_a[i]*mas_b[i-1]          # Третя координата
dovgina = math.sqrt ((c1)*(c1)+(c2)*(c2)+(c3)*(c3))   # Довжина вектора
print ('c ','(',c1,';',c2,';',c3,') ')               # Виведення координат вектора
print ('довжина = ', dovgina)                        # Виведення довжини вектора
```

Рис. 10.7. Програма обчислення координат і довжини векторного добутку

Результат виконання програми наведено зліва.

```
координата a( 0 )
2
координата b( 0 )
-4
координата a( 1 )
-3
координата b( 1 )
5
координата a( 2 )
7
координата b( 2 )
1
c  ( -38.0 ; -30.0 ; -2.0 )
довжина =  48.45616575834287
```

Запитання для перевірки знань

1. Які операції можна виконувати над векторами?
2. Сформулюйте правило додавання двох векторів на площині.
3. Сформулюйте правило віднімання двох векторів на площині.
4. Як виконати в координатах операцію складання векторів?
5. Сформулюйте правило знаходження скалярного добутку векторів.
6. Що називають векторним добутком?
7. Сформулюйте правило знаходження векторного добутку в координатах.

Завдання для самостійного виконання

1. Знайдіть координати вектора $\overrightarrow{AB}$, якщо $A(-2;3)$, $B(2;-4)$.
2. Скалярний добуток вектора $\vec{a}$ на вектор $\vec{b}$ дорівнює 2, а їх модулі: $|\vec{a}|=2$, $|\vec{b}|=2$. Знайдіть кут між цими векторами.
3. Обчисліть скалярний добуток векторів $\vec{a}$ та $\vec{b}$, якщо їхні довжини відповідно дорівнюють 3 і 4, а кут між ними становить 60°.
4. Знайдіть векторний добуток векторів: $\vec{a}=(6;7;9)$ і $\vec{b}=(8;5;10)$.
5. Виконайте програму на рис. 10.7 для векторів, дані про які наведені в завданні 4. Доведіть, що результат отримано правильний.

10.3. Обчислення площі многокутника

Які фігури в геометрії називають многокутниками? Пригадайте, як ви обчислювали площі ромба, прямокутника, паралелограма та інших фігур.

Розв'язування багатьох геометричних задач пов'язано з необхідністю обчислення площі многокутника, заданого координатами його вершин. У свою чергу, визначення площі многокутника ґрунтується на обчисленні площі трикутника, для чого зручно користуватися *орієнтованим кутом* між векторами, який враховує взаємне розташування векторів.

Орієнтованим кутом між векторами $\vec{a}$ і $\vec{b}$ називають кут, на який слід повернути вектор $\vec{a}$ проти ходу годинникової стрілки, щоб він став співнапрямленим з вектором $\vec{b}$.

За абсолютним значенням орієнтований кут дорівнює звичайному куту між векторами, але він є додатним або від'ємним. Орієнтований кут між векторами $\vec{a}$ і $\vec{b}$ є додатним, якщо обертання від вектора $\vec{a}$ до вектора $\vec{b}$ здійснюється проти ходу годинникової стрілки, і від'ємним — у протилежному напрямку.

Орієнтовані кути, що відрізняються на 360°, вважаються **рівними**.

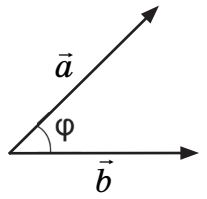


Рис. 10.8. Кут між векторами

Наприклад, на рис. 10.8 орієнтовані кути між векторами $\vec{a}$ і $\vec{b}$ і між векторами $\vec{b}$ і $\vec{a}$ за модулем рівні, але перший із них від'ємний, а другий — додатний. Таким чином, величина орієнтованого кута залежить від порядку переліку векторів і може набувати значень від $-180°$ до $180°$.

Розглянемо тепер поняття орієнтованої площі трикутника.

Трикутник називають орієнтованим, якщо зазначено напрям обходу його контуру.

Орієнтованою площею трикутника ABC називають величину, що дорівнює його площі, взятій зі знаком «плюс», якщо обхід сторін трикутника у порядку $A–B–C–A$ здійснюється проти ходу годинникової стрілки, і зі знаком «мінус», якщо — за ходом.

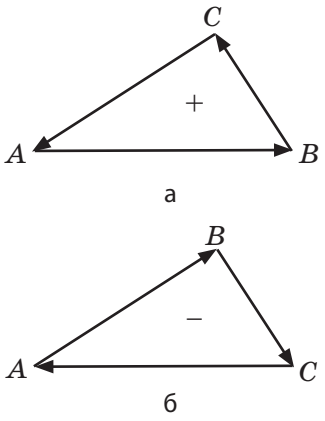


Рис. 10.9. Варіанти обходу вершин трикутника

Наприклад, площа трикутника, зображеного на рис. 10.9, *а*, при обході його вершин $A–B–C–A$ є додатною, а на рис. 10.9, *б*, при обході вершим $A–B–C–A$ — від'ємною. Отже, знак орієнтованої площі трикутника залежить від порядку обходу вершин.

Площу трикутника можна обчислити в такий спосіб.

Розглянемо прямокутний трикутник ABC (рис. 10.10). Нехай O — довільна точка на площині трикутника. Площу трикутника ABC можна знайти так: від площі трикутника OBC відняти площі трикутників OAB і OCA. Інакше кажучи, необхідно додати орієнтовані площі трикутників OAB, OBC і OCA. Це правило діє у випадку будь-якого вибору точки O.

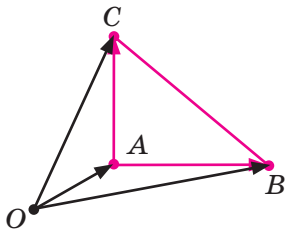


Рис. 10.10. Приклад обчислення орієнтованої площі трикутника

Перейдемо тепер до визначення площі многокутника. Для обчислення площі будь-якого многокутника $A_1A_2\ \ldots\ A_n$ необхідно скласти орієнтовані площі трикутників OA_1A_2, OA_2A_3, ..., OA_nA_1.

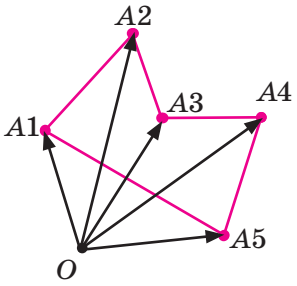


Рис. 10.11. П'ятикутник $A_1A_2A_3A_4A_5$

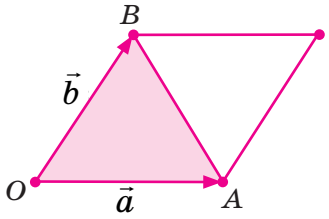


Рис. 10.12. Площа паралелограма, побудованого на двох векторах

Величину $x_1y_2 - x_2y_1$ називають косим (псевдоскалярним) добутком векторів $\vec{a}$ і $\vec{b}$. Для нього будемо використовувати позначення $[\vec{a}, \vec{b}]$( ), що є скалярною величиною і має таку властивість: $[\vec{a}, \vec{b}] = -[\vec{b}, \vec{a}]$.

Наприклад, для обчислення площі п'ятикутника $A_1A_2A_3A_4A_5$, зображеного на рис. 10.11, необхідно скласти орієнтовані площі трикутників: OA_1A_2, OA_2A_3, OA_3A_4, OA_4A_5, OA_5A_1.

Площа многокутника буде зі знаком «плюс», якщо обхід вершин A_1A_2 ... A_n многокутника здійснюється проти ходу годинникової стрілки, і зі знаком «мінус», якщо обхід виконується за ходом годинникової стрілки. Ця площа називається орієнтованою площею многокутника A_1A_2 ... A_n.

Отже, обчислення площі многокутника фактично звелося до знаходження орієнтованої площі трикутника. Виразимо її в координатах. Орієнтована площа, побудована на векторах $\vec{a} = (x_1; y_1)$ і $\vec{b} = (x_2; y_2)$, є площею паралелограма (рис. 10.12).

Векторний добуток, виражений через координати векторів, визначають так:

$$[\vec{a}, \vec{b}] = \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} = x_1y_2 - x_2y_1.$$

Площа трикутника дорівнює половині цієї площі:

$$S = \frac{1}{2}(x_1y_2 - x_2y_1).$$

Точку O зручно брати як початок координат. У такому випадку координати векторів, на основі яких обчислюються орієнтовані площі, збігаються з координатами точок. Нехай $(x_1; y_1)$, $(x_2; y_2)$, ..., $(x_N; y_N)$ — координати вершин заданого многокутника у порядку обходу за ходом годинникової стрілки або проти її ходу. Тоді його орієнтована площа S дорівнює:

$$S = \frac{1}{2}\left[(x_1 \cdot y_2 - x_2 \cdot y_1) + (x_2 \cdot y_3 - x_3 \cdot y_2) + \ldots + (x_n \cdot y_1 - x_1 \cdot y_n)\right].$$

Цю формулу можна записати в такому вигляді:

$$S = \frac{1}{2}\left(x_1(y_2 - y_n) + x_2(y_3 - y_1) + x_3(y_4 - y_2) + \ldots + x_n(y_1 - y_{n-1})\right).$$

Нагадаємо, що звичайний кут дорівнює модулю орієнтованого кута. Відзначимо також, що усі розглянуті питання справедливі для правої системи координат. Але для окремих задач зручніше застосовувати ліву систему координат.

Наприклад, координати пікселів на екрані монітора подають у лівій системі координат (вісь абсцис напрямлена вправо, вісь ординат — униз). У випадку вибору таких осей додатним є кут повороту за ходом годинникової стрілки. З цією поправкою все сказане раніше застосовується і до лівої системи координат.

Алгоритм обчислення площі многокутника за вже отриманими формулами може бути різним. Зверніть увагу на те, що логіка обчислення першого й останнього членів цього виразу відрізняється від логіки обчислення всіх інших членів. Тому обчислимо окремо перший і останній члени, а всі інші обчислюватимемо за тією самою схемою.

Програму обчислення площі многокутника на основі такого підходу зображено на рис. 10.13. У програмі обхід вершин починається з вершини A_0 і здійснюється в порядку збільшення їх номерів.

```
# Обчислення площі многокутника
n = int(input ('Кількість вершин '))    # Кількість вершин многокутника
mas_x = []                              # Порожній список координат x
mas_y = []                              # Порожній список координат y
for i in range (3):                     # Цикл уведення координат многокутника
    print ('Координата x(', i,') ')     # Повідомлення про уведення координати x
    a = float(input())                  # Уведення чергової координати x
    mas_x.append (a)                    # Формування списку координат x
    print ('Координата y(', i,') ')     # Повідомлення про уведення координати y
    a = float(input())                  # Уведення чергової координати y
    mas_y.append (a)                    # Формування списку координат y
S3 = 0                                  # Початкове значення частки площі
for i in range (n-1):                   # Циклічний обхід вершин многокутника
    if i == 0:                          # Якщо обрана початкова вершина
        S1 = (mas_x[0])*(mas_y[1]-mas_y[n-1])       # Площа для початкової вершини
    if (i>0) and (i<n-1):               # Якщо вершина не початкова і не остання
# Обчислення площ для всіх вершин, крім початкової й останньої
        a = (mas_x[i])*(mas_y[i+1]-mas_y[i-1])
        S3 = S3+a                       # Сума площ, крім початкової й останньої вершини
    else:                               # Якщо обрана остання вершина
        S2 = (mas_x[n-1])*(mas_y[0]-mas_y[n-2])      # Площа для останньої вершини
S = abc(S1+S2+S3)                       # Обчислення загальної площі многокутника
print ('S = ', S/2)                     # Виведення загальної площі многокутника
```

Рис. 10.13. Програма обчислення площі многокутника

На рис. 10.14 наведено динаміку введення значень координат вершин правильного шестикутника і результат обчислення його площі:

```
Кількість вершин 6
Координата x( 0 )
3
Координата y( 0 )
6
Координата x( 1 )
3
Координата y( 1 )
10
Координата x( 2 )
6.42
Координата y( 2 )
12
Координата x( 3 )
9.84
Координата y( 3 )
10
Координата x( 4 )
9.84
Координата y( 4 )
6
Координата x( 5 )
6.42
Координата y( 5 )
4
S = 41.04
```

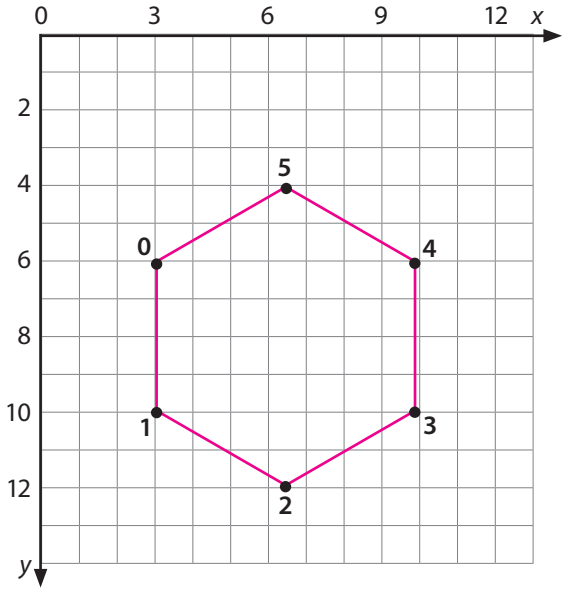


Рис. 10.14. Результат виконання програми — многокутник за заданими координатами

Запитання для перевірки знань

1. Поясніть сутність орієнтованого кута.
2. У якому випадку орієнтований кут має додатне значення?
3. Поясніть сутність орієнтованої площі трикутника.
4. Наведіть приклад обчислення орієнтовної площі трикутника.
5. Що називають орієнтованою площею многокутника?
6. Поясніть порядок обчислення площі многокутника за допомогою орієнтованих площ трикутників.
7. За якою формулою обчислюють площу многокутника?

Завдання для самостійного виконання

1. Знайдіть площу трикутника, вершини якого мають такі координати:
$A(1;-3)$, $B(2;2)$, $C(-5;4)$.
2. Обчисліть площу чотирикутника, вершини якого мають координати:
$A(3;3)$, $B(9;5)$, $C(11;8)$, $D(2;6)$.
3. Вершини п'ятикутника мають координати:
$A_1(0.6;2.1)$, $A_2(1.8;3.6)$, $A_3(2.2;2.3)$, $A_4(3.6;2.4)$, $A(3.1;0.5)$.
Обчисліть площу цього п'ятикутника.
4. Виконайте програму, зображену на рис. 10.13, для трикутника, що має вершини: $A(-1;2)$, $B(5;3)$, $C(3;4)$.
Доведіть, що отриманий результат є правильним.
5. Модифікуйте програму, зображену на рис. 10.13, так, щоб значення координат вершин многокутника не вводилися з клавіатури, а їх значення визначалися безпосередньо всередині програми. Перевірте програму на прикладі прямокутника і доведіть, що вона виконується правильно.
6. Одна з вершин правильного шестикутника лежить у початку системи координат. Розробіть програму обчислення площі шестикутника з урахуванням цієї обставини.
7. Скористайтеся географічною картою України і визначте, яка фігура утвориться, якщо сполучити відрізками міста: Львів, Чернігів, Полтава, Дніпро, Херсон, Одеса, Чернівці, Львів. Знайдіть площу цієї фігури (рис. 10.15).

Рис. 10.15. Карта України з обласними центрами

10.4. Побудова опуклої оболонки

Чи доводилося вам використовувати оболонки в процесі вивчення інших навчальних предметів та в повсякденному житті? Якщо доводилося, то які типи оболонок ви використовували?

На рис. 10.16 подано три варіанти розташування скінченної множини точок. Усі вони розташовані всередині замкненої лінії, або на цій лінії, або є вершинами замкненої ламаної. Кожна з цих ліній не має самоперетинів.

Дотична, проведена до кривої на рис. 10.16, *а*, може перетинати її в кількох точках, тому вона не є опуклою, її часто називають просто оболонкою.

Криву лінію, наведену на рис. 10.16, *б*, називають опуклою оболонкою, оскільки будь-яка дотична до цієї лінії більше не перетинає її в жодній точці.

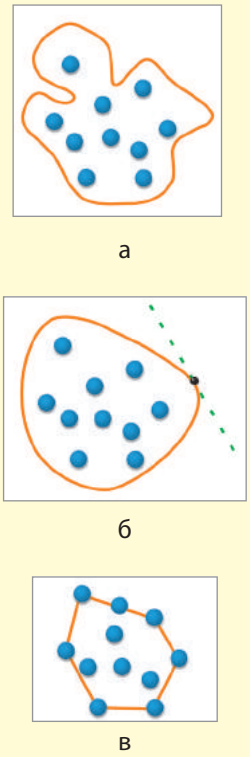
Опуклу оболонку, наведену на рис 10.16, *в*, називають оболонкою мінімальної довжини, або просто мінімальною опуклою оболонкою. Точки множини цієї оболонки лежать або на її вершинах, або на прямих, що сполучають відповідні вершини. Далі розглядатимемо тільки мінімальні опуклі оболонки, які стисло називатимемо *опуклими оболонками*.

Отже, опуклою оболонкою скінченної множини точок K називають найменший опуклий многокутник, що містить усі точки множини K, деякі з яких лежать усередині многокутника, деякі є його вершинами, а деякі лежать на його сторонах.

Кожна точка на площині має свої координати. Наприклад, точка p_0 має координати x_0y_0, точка p_1 — координати x_1y_1, а точка p_i — координати x_iy_i. Завдання побудови мінімальної опуклої оболонки зводиться до відбору з множини K таких точок, що є вершинами многокутника, за умови, що всі інші точки цієї множини лежать або в його середині, або на сторонах многокутника.

Існують різні алгоритми побудови мінімальної опуклої оболонки, одним із найпростіших є алгоритм Джарвіса.

Сутність алгоритму Джарвіса така.



а б в

Рис. 10.16. Варіанти оболонок

Крок 1

Із множини точок K обирається стартова точка, що гарантовано входить до складу вершин опуклої оболонки.

Найчастіше вибирають найнижчу праву або ліву точку.
На рис. 10.17 найнижчою є точка p_0. Якби було декілька точок, розташованих на одній найменшій висоті, то вибрали б ту, що справа.

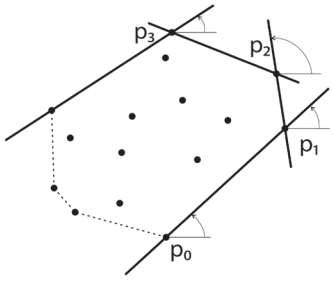


Рис. 10.17. Точки на площині

Крок 2

Здійснимо побудову опуклої оболонки.

Щоб знайти наступну точку від точки p_0, яка буде черговою вершиною оболонки, проводимо від неї вправо горизонтальну лінію і повертаємо її навколо точки p_0 проти ходу годинникової стрілки доти, поки вона не торкнеться точки p_1 із множини точок *K*. Відрізок $\overrightarrow{p_0p_1}$ буде ребром опуклої оболонки.

Аналогічні дії виконуємо в точці p_1, знаходимо точку p_2 і ребро $\overrightarrow{p_1p_2}$. Цей процес продовжується доти, поки не повернемося у точку p_0.

Очевидно, що точка p_1 має таку властивість, що всі інші точки лежать ліворуч від вектора $\overrightarrow{p_0p_1}$. Це означає, що орієнтований кут між вектором $\overrightarrow{p_0p_1}$ і всіма іншими векторами $\overrightarrow{p_0p_i}$ невід'ємний для будь-якої точки i з усієї множини точок. Знайдемо точку p_1. Для цього слід:

– обчислити косий добуток $\left[\overrightarrow{p_0p_1}, \overrightarrow{p_1p_m}\right]$;

– перевірити умову $\left[\overrightarrow{p_0p_1}, \overrightarrow{p_1p_m}\right] \geqslant 0$ для всіх точок *m*, де *m* — усі точки, крім нульової і першої.

Зрозуміло, що цій умові на рис. 10.17 відповідає тільки точка p_1.

Для інших варіантів розміщення точок цій умові може відповідати декілька точок. Це означає, що на векторі p_0p_1 може бути розташовано декілька точок. У такому випадку вибирають ту з них, для якої довжина вектора $\overrightarrow{p_0p_1} = (x_1 - x_0; y_1 - y_0)$ є максимальною.

Так само слід діяти, якщо вказана умова виконується для інших точок.

Якщо умова $\left[\overrightarrow{p_0p_1}, \overrightarrow{p_1p_m}\right] \geqslant 0$ не виконується, то це означає, що точка p_1 не може бути вершиною опуклої оболонки і слід перейти до чергової точки p_2.

Пояснимо реалізацію алгоритму Джарвіса на прикладі.

Американська розробниця ПЗ Радія Джой Перлман стала першою, хто почав навчати програмуванню дітей молодшого віку. Вона розробила дитячу версію навчальної робототехнічної мови LOGO (названу TORTIS).

Приклад. Нехай на площині розміщено 7 точок із такими координатами:

Номер точки	0	1	2	3	4	5	6
Координата *x*	5	3	9	9	13	3	14
Координата *y*	7	4	6	15	2	2	10

Якщо ці точки нанести на площину, то можна переконатися, що права нижня точка розташована на четвертій позиції. Оскільки ця точка є стартовою, то для зручності розроблення програми доцільно поміняти місцями координати цієї точки і точки, розташованої на першій позиції.

У результаті будемо мати таке розташування точок:

Номер точки	0	1	2	3	4	5	6
Координата *x*	13	3	9	9	5	3	14
Координата *y*	2	4	6	15	7	2	10

Програму реалізації алгоритму Джарвіса зображено на рис. 10.18.

```
# Програма реалізації алгоритму Джарвіса
n = int(input('Кількість точок: '))                    # Введення кількості точок
x = [10, 5, 11, 13, 7, 3, 17, 8];
y = [14, 16, 4, 17, 7, 3, 9, 10]                       # Координати точок
t = [0]*n; xn = [0]*n; yn = [0]*n                      # Додаткові порожні списки
# Пошук у списку x значення мінімальної координати
m = 0                       # Перший елемент списку x приймається за мінімальний
for i in range (1, n):      # Цикл пошуку лівої крайньої точки
    if x[i]<x[m]:           # Якщо чергова координата x менша від поточної мінімальної
        m = i               # Запам'ятовується позиція з мінімальною координатою x
    else:                   # Якщо у списку є декілька точок з однаковою мінімальною
# координатою x, то обирається нижня ліва точка
        if (x[i] == x[m]) and (y[i] < y[m]):
            m = i
k = 0                       # Початкове значення кількості точок оболонки
x1 = x[m]; y1 = y[m]        # Запам'ятовування координат x і y, розташованих на позиції m
xn[k] = x1; yn[k] = y1      # На позиції 0 списків xn і yn координати лівої точки
# Побудова мінімальної опуклої оболонки
poisk = True                # Ознака продовження пошуку точки оболонки
while poisk:                # Доти, доки ознака poisk має значення True
    poisk = False           # Ознака закінчення пошуку точки оболонки
    for i in range (n):     # Цикл побудови оболонки
        if t[i] == 0:       # Якщо точка ще не розглядалася
            x2 = x[i]; y2 = y[i]              # x2 і y2 – допоміжні змінні
            pt = True                         # Ознака косого добутку
            for j in range (n):               # Цикл побудови оболонки
                if (x[j]-x1)*(y2-y1)-(y[j]-y1)*(x2-x1)<0:      # Кос. добуток > 0?
                    pt = False                # Косий добуток додатний
                if not (pt):                  # Якщо ознака pt має значення False
                    for j in range (n):       # Цикл побудови оболонки
                        if (x[j]-x1)*(y2-y1)-(y[j]-y1)*(x2-x1)<0:     # Кос. доб. < 0?
                            pt = False        # Ознака від'ємного косого добутку
            if pt:                            # Якщо ознака pt має значення True
                k = k+1                       # Збільшення кількості точок оболонки
                x1 = x2; y1 = y2              # x1 і y1 – допоміжні змінні
                t[i] = k                      # У список додається черговий номер точки
                xn[k] = x1; yn[k] = y1        # Запам'ятовуються координати точки оболонки
                poisk = True                  # Ознака продовження пошуку точок оболонки
print ('Координати точок опуклої оболонки')    # Повідомлення
for i in range (k):                           # Цикл виведення
    print (xn[i],'',yn [i])                   # Виведення координат точок оболонки
```

Рис. 10.18. Програма реалізації алгоритму Джарвіса

Результат виконання програми:

```
Кількість точок: 8
Координати точок опуклої оболонки
3  3
11  4
17  9
13  17
5  16
```

Запитання для перевірки знань

1. Які оболонки називають опуклими?
2. Які опуклі оболонки називають мінімальними?
3. Накресліть мінімальну опуклу оболонку.
4. Яку точку вибирають, якщо умові «косий добуток» відповідають декілька векторів?
5. Сформулюйте сутність алгоритму Джарвіса.

Завдання для самостійного виконання

1. Побудуйте мінімальну опуклу оболонку для точок із такими координатами:

X	2	3	4	5	5	7	10
Y	5	7	2	6	5	1	8

2. Скористайтеся географічною картою України і зобразіть точками у прямокутній системі координат міста: Київ, Харків, Львів, Одеса, Черкаси, Житомир, Херсон, Запоріжжя, Полтава, Чернігів, Луцьк, Ужгород. Побудуйте на основі цих міст мінімальну опуклу оболонку. Координати яких міст не розташовані на ребрах і вершинах оболонки?
3. Виконайте програму, зображену на рис. 10.16, для власного набору точок. Доведіть, що програма виконана правильно.
4. Модифікуйте програму, зображену на рис. 10.16, так, щоб обхід вершин опуклої оболонки починався з верхньої лівої точки.
5. На площині в прямокутній системі координат розташовано n точок ($n>4$), з відомими координатами. Чотири з цих точок є вершинами квадрата, одна зі сторін якого паралельна осі абсцис. Розробіть програму визначення координат точок, які містяться всередині квадрата.

Тест 5 Тест 6 Тест 7

Тест 8 Тест 9 Тест 10

Тестові завдання з автоматичною перевіркою результату на сайті interactive.ranok.com.ua

Розділ 3. ВЕБ-ТЕХНОЛОГІЇ

11.1. Основні тренди у веб-дизайні

Якими сайтами ви користуєтесь? Що вас у них приваблює?

Успіх сайта значною мірою залежить від якості дизайну, отже, дуже важливо стежити за трендом. Потрібно розуміти, які інструменти ввійшли в моду в сфері сайтобудування, а які вже вважаються застарілими і від яких варто відмовитися. Адже тренди у веб-дизайні постійно змінюються, удосконалюються, модернізуються.

Загальна кількість користувачів мережі Інтернет постійно збільшується. Розглянемо статичні дані, наведені на рис. 11.1. Упродовж останніх років незмінними залишаються дві тенденції: необхідність швидкого завантаження сайтів та орієнтація на мобільні версії сайтів.

Тренд (від англ. *trend* — напрям, тенденція) — можливий (ймовірний) вектор розвитку подій стосовно якогось проміжку часу.

- Швидкість завантаження сайтів

Під час завантаження сайта саме швидкість має вирішальне значення. У ході досліджень доведено, що людині достатньо трьох секунд для того, щоб справити враження на оточуючих. Це стосується й сайта: якщо за цей час він так і не завантажиться, велика ймовірність, що відвідувач далі не чекатиме.

- Орієнтація на мобільні сайти

85 % користувачів впевнені, що на смартфоні сайт повинен виглядати не гірше (а то й краще), ніж на моніторі стаціонарного комп'ютера чи ноутбука.

Рис. 11.1. Статистичні дані користування Інтернетом у 2019 році за даними Аналітичної агенції **We Are Social** і SMM-платформи **Hootsuite**

Щодня аудиторія Інтернету збільшується на 1 млн нових користувачів.
Усе більше користувачів для серфінгу Інтернетом використовують саме смартфони.

Джош Кларк, фахівець у галузі мобільного дизайну, «дизайнер взаємодії», як він себе називає, який вдало поєднує ергономіку та психологію мобільного дизайну, у книзі «Проектування для дотику» (*Josh Clark, Designing for Touch*) досліджує, як користувачі тримають свої мобільні телефони та як їхні рухи мають оброблятися в процесі роботи з сайтом.

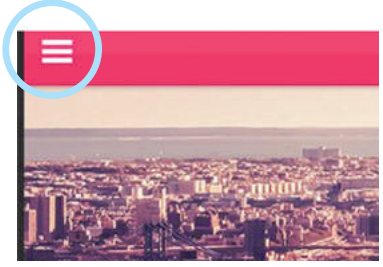


Рис. 11.2. Приклад меню **гамбургер**

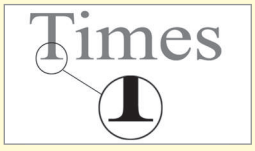


Рис. 11.3. Приклад шрифтів із зарубками

Рис. 11.4. Приклад шрифтів без зарубок

Розглянемо напрями, які на початок 2019 року є найбільш популярними за версіями інтернет-журналів AWWWARD і Designmodo.

- Візуалізація даних — інфографіка сприймається як частина дизайну. Візуалізація даних є цікавим і наочним способом подання великих обсягів інформації.

Цей формат є дуже популярним серед користувачів, адже його можна застосовувати всюди: від чисел і карт до складних алгоритмів. Рис. 11.1 є яскравим прикладом подібної інфографіки.

- Персоналізація сайтів — найуспішнішими стануть сайти, орієнтовані на користувача, тобто ті, які надають інформацію з урахуванням статі, віку і навіть особистих уподобань користувача. Відповідно електронна комерція стане ще більш витонченою.
- Зростання ролі типографіки — зараз у тренді незвичайні шрифти, різноманітні художні елементи, креативні прийоми на кшталт складання текстів з «газетних вирізок». З'явився термін «вінтажна типографіка» (стиль старовинної типографії).

Мобільний перегляд впевнено випереджає десктопний, і загальний дизайн сайтів стає все більш зручним для пальців. Все частіше доведеться стикатися з навігацією, пристосованою до великого пальця. Так, меню «гамбургер» (рис. 11.2), яке традиційно розташовується нагорі сайтів, буде перенесено в нижню частину мобільних екранів.

Серед веб-дизайнерів набуває популярності використання шрифтів із зарубками (serif) (рис. 11.3), заокруглених плит і текстових елементів, які здаються більш старими.

Свого часу вважалося, що більш читабельними на екранах є шрифти без зарубок (рис. 11.4). Проте завдяки еволюції девайсів вибір шрифтів перестає бути проблемою. Набувають популярності так звані варіативні шрифти. Змінні літери, які поступово збільшуються і зменшуються, привабливі для користувачів.

- Пріоритетність відео-контенту — на YouTube щодня переглядають близько 500 млн годин відео.

При цьому передбачається, що в 2019 році на призначений для користувача контент припадатиме 80 % усього інтернет-трафіку. На лідируючі позиції серед інших компонентів веб-дизайну вийдуть повноекранні відео, оскільки вони є інформативними та зручними.

- Синемаграфіка — ці статичні картинки містять один анімований (динамічний) елемент. Наприклад, це може бути статична композиція з паруючою чашкою кави.

Таке поєднання фотографії та анімації є особливо привабливим з точки зору маркетологів.

- Геометричні форми (рис. 11.5) — використання у веб-дизайні різних композицій геометричних форм було започатковано в 2016 році, і, за прогнозами, у 2019-му лише набиратиме популярності.
- 3D-графіка (рис. 11.6) — актуальний тренд, який можна назвати природною еволюцією плоского дизайну.

Тривимірні візуалізації забезпечують оновлення такого дизайну. Результатом є поєднання 3D-реалістичних і плоских інтерфейсів, які є складними та візуально цікавими. Вони «чіпляють» і привертають пильну увагу користувача.

Рис. 11.5. Використання геометричних форм у веб-дизайні

Рис. 11.6. Приклад тривимірної графіки

- Яскраві кольори. Це тренд, який ніколи не втрачає актуальності. Використання градієнту є багатофункціональною кольоровою тенденцією, що працює в дизайну будь-якого типу. Це може бути цікавим способом розбити елементи тексту або виділити певний контент, або створити неординарний фон для розміщення продукту. З точки зору колористики застосування градієнтів (рис. 11.7) є чудовою ідеєю. Фон сторінки, у створенні якого задіяний градієнт, викликає враження унікальності і свіжості.
- Застосування розділеного екрана (спліт-екрана). Ми неодноразово наголошували, що дизайн сайтів має орієнтуватися на мобільних користувачів. Відповідно з'являються шаблони розділених екранів. Естетичний поділ екрана (насамперед за рахунок кольору) передбачає, що сайт, призначений для перегляду на великому моніторі,

Нині все більшої популярності набуває візуальний пошук. Його дозволяють використовувати **Snapchat**, **Pinterest** (соціальний фото-сервіс), **Lens**, **AliExpress** і **Google Lens**. Піонером у візуальному пошуку був **Amazon** із його функцією **Flow**. У 2014 році він використовував розпізнавання зображень для пошуку товару.

У вересні 2018 року **Snapchat** (мультимедійний мобільний додаток обміну фото та відео) у співробітництві з **Amazon** оголосив про розробку нової функції. Вона дозволяє користувачеві «сканувати» штрих-коди за допомогою камери.

Google Lens — застосунок для визначення об'єктів в об'єктиві камери. Він може виділяти назви в тексті і навіть телефони, відразу дозволяючи зателефонувати за вказаним номером. Він розпізнає більш ніж 1 млрд об'єктів

Рис. 11.7. Приклад використання градієнта

розподілено на дві панелі, які на екранах менших пристроїв згортаються у вертикальні блоки. Це простий спосіб адаптивного подання контенту (див. матеріал §11.10). Переважна більшість веб-дизайнерів переходять на асиметричне розбиття для змісту.

Подібне розбиття екрану передбачає певний взаємозв'язок контенту — він доповнює або контрастує. Популярною є схема, коли під час наведення миші на одну частину сторінки затінюється решта, що дає змогу фокусуватися на вибраному (рис. 11.8).

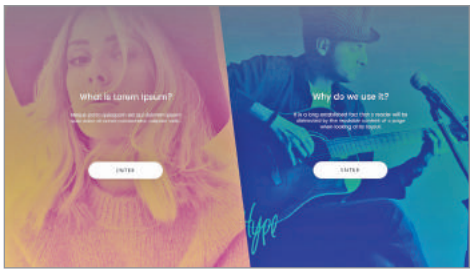


Рис. 11.8. Спліт-екран

- Поширення використання штучного інтелекту. Штучний інтелект так чи інакше вже використовують Amazon, Netflix та ін. Насамперед це стосується чат-ботів, які надають потрібну інформацію, наприклад, під час обслуговування клієнтів (рис. 11.9).

Із зростанням кількості користувачів, які використовують смарт-пристрої у своїх будинках, і смартфонів для управління цими пристроями переважна більшість власників надає перевагу голосовим командам. Як наслідок зростає роль голосового інтерфейсу, налаштованого власником за допомогою ключових слів і відповідних дій.

Голосові інтерфейси базуються на «читанні» веб-сайтів для отримання інформації та даних. Таким чином, все більше уваги приділяється розробці сайтів, здатних взаємодіяти з голосовими інтерфейсами.

Чат-бот (англ. *Chatbot*) — комп'ютерна програма, розроблена на основі нейромереж та технологій машинного навчання, яка веде розмову за допомогою слухових або текстових методів. Програма імітує розмову з людиною в Інтернеті, саме тому цей сервіс найкраще зарекомендував себе в месенджерах (Facebook Messenger, Telegram тощо).

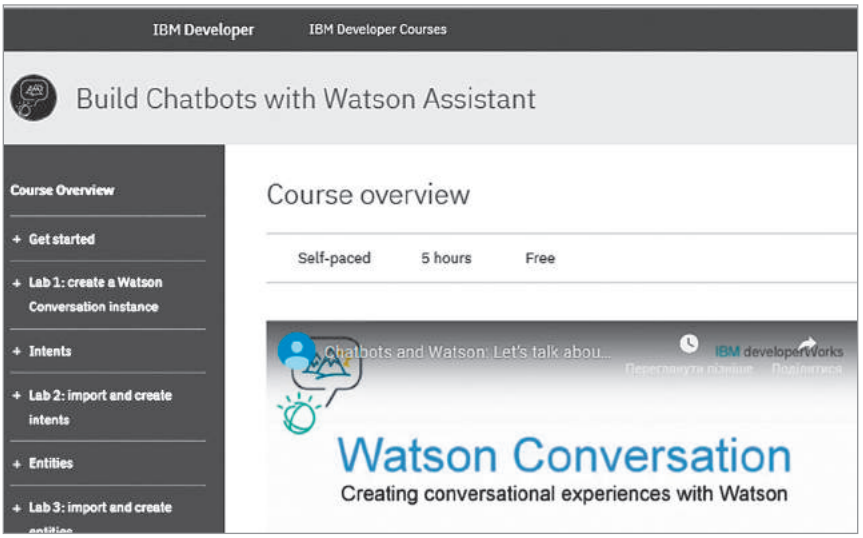


Рис. 11.9. Приклад створення чат-боту засобами IBM Watson Assistans

Запитання для перевірки знань

1. Що таке чат-бот? Де використовують чат-боти?
2. Що таке синематографіка?
3. Наведіть приклади використання штучного інтелекту в сучасних онлайн-платформах.
4. У чому перевага розподілених екранів?
5. Знайдіть в Інтернеті статистику за 2017 та 2018 роки, аналогічну наведеній на рис. 11.1. Яку тенденцію ви можете простежити?
6. Які, на вашу думку, тренди будуть актуальними в найближчому майбутньому?

11.2. Види сайтів та цільова аудиторія

Зробіть невеличке опитування, з'ясуйте, які сайти користуються найбільшою популярністю серед ваших друзів.

У березні 2019 року Всесвітня павутина відсвяткувала своє тридцятиріччя. Тім Бернерс Лі, один з головних її розробників, запропонував перший веб-сервер, перший браузер (більш детально в §11.7) і редактор (*the «WorldWideWeb.app.»*). Наразі кількість сайтів перевищує півтора мільярда і продовжує невпинно зростати. Щосекунди у світі з'являються два сайти (рис. 11.10).

Ознайомимося з класифікацією сайтів (рис. 11.11).

Інформаційні сайти є одними з найбільш популярних у мережі Інтернет і призначені для донесення до користувача будь-якої інформації.

До інформаційних сайтів відносять тематичні інформаційні, новинні та блоги.

Тематичні інформаційні сайти — ресурси, в яких практично всі сторінки присвячені одній певній проблематиці або темі, у свою чергу різнотематичні сайти охоплюють широке коло інформаційної спрямованості і можуть висвітлювати велику кількість тематик і напрямків.

Новинні сайти виконують дуже важливе завдання — вони повинні донести до користувача різні новини, що відбувалися, відбудуться чи відбуваються в цей момент. Такі сайти можуть спеціалізуватись на одній певній тематиці або розповідати інтернет-спільноті про новини з різних життєвих сфер людини.

Блоги зустрічаються у Всесвітній павутині досить часто, рівень їхньої популярності стає дедалі вищим. Як і новинні сайти, блоги призначені для того, щоб донести інтернет-спільноті ту чи іншу інформацію, але з однією істотною відмінністю: автор описує свою особисту думку про те, що відбувається.

Умовно блоги діляться на дві категорії: особисті та корпоративні, кожна з яких виконує певну функцію.

Особисті блоги — категорія сайтів, у яку входять ресурси, створені окремими користувачами, з метою донести широкому колу громадськості думку автора з певної теми.

Корпоративні блоги зазвичай створюються певними компаніями як додатковий ресурс, який розкручує бренд, і часто є доповненням до комерційного або корпоративного сайта.

Залежно від того, яку зі згаданих проблем має вирішувати корпоративний ресурс, сайти поділяють на іміджеві та інформаційні. *Іміджевий корпоративний сайт* виконує одну з найважливіших функцій для компанії — рекламну. На ньому

Дізнатись кількість існуючих сайтів можна завдяки проекту **Internet Live Stats** (http://www.internetlivestats.com/total-number-of-websites/).

Рис. 11.10. Кількість сайтів на сьогодні

Рис. 11.11. Класифікація сайтів

Корпоративні блоги створюються організаціями або компаніями для виконання одного з двох певних завдань: зміцнення зв'язку між філіями та відділеннями або підвищення іміджу в очах потенційних та існуючих клієнтів.

зазвичай описується історія підприємства, компанії або торгової марки, надаються докладні поточні відомості, контактна інформація, містяться відомості про вироблювані і розповсюджувані товари, про надавані послуги. *Інформаційний корпоративний сайт* виконує не менш значущі функції — автоматизація та зміцнення зв'язку між відділами та філіями, обіг документів, управління персоналом тощо.

На **іміджевих сайтах** можуть публікуватися різні новини, інформація про акції, знижки — все те, що характеризує компанію перед клієнтом. Такі сайти відрізняються оригінальним ексклюзивним дизайном, поєднанням нестандартних елементів оформлення і рішень.

Сайти-портфоліо і сайти-візитки сьогодні все частіше створюються приватними особами, особливо фрілансерами, які намагаються привернути увагу нових замовників і клієнтів, розрекламувавши свою роботу і надавши результат її виконання.

Сайти-візитки призначені для того, щоб коротко подати інформацію про свого власника широкому колу користувачів. Такі сайти налічують більше ніж 20 сторінок.

Сайти-портфоліо призначені привертати увагу широкої цільової аудиторії шляхом рекламування роботи, виконаної компанією, щоб клієнт зміг вирішити, чи потрібно користуватися послугами цієї компанії.

Комерційні сайти — одна з найбільш поширених категорій сайтів. Їх основне призначення — продаж товарів інтернет-користувачам.

Пересічні користувачі Інтернету зазвичай не мають відкритого доступу до інформаційного корпоративного сайта, оскільки він призначений виключно для співробітників однієї організації.

Комерційні сайти поділяють на кілька підкатегорій, найбільш поширеними з яких є сайт-вітрина, промо-сайт, інтернет-магазин.

Сайти-вітрини, або *посадкові сторінки* (*landing page*) — сайти, основним призначенням яких є не продаж готової продукції, а її реклама. Найбільшого поширення вони отримали серед компаній і підприємств, які виробляють товари.

За допомогою **сайта-вітрини** неможливо здійснити операцію купівлі-продажу, оскільки він лише надає докладну інформацію про товар. Але іноді, на них зазначаються місця, де можна купити товар, який зацікавив.

Промо-сайти — інтернет-ресурси, на яких рекламують певні послуги або товари, зупиняючи особливу увагу на їхніх перевагах. Вони характеризуються досить простою структурою, невеликим обсягом (до 10 сторінок) і нетривалим терміном існування.

На сторінках промо-сайта містяться різні графічні матеріали, контактні дані компанії тощо. Якщо необхідно провести масштабну компанію певного товару — промо-сайт стане чудовим помічником.

Інтернет-магазини призначені для продажу товарів різних категорій у мережі в режимі онлайн. Відвідавши такий сайт, кожен користувач Всесвітньої павутини може вибрати товар що його цікавить, дізнатися про нього докладну інформацію і, в разі необхідності, здійснити замовлення.

Дуже часто компанії пропонують користувачам усі свої лінійки товарів на одному сайті. Значно рідше створюються кілька ресурсів, на яких представлено одну певну категорію поширюваної продукції.

Соціальні проекти включають форуми, спеціалізовані соціальні мережі, мережі загальної спрямованості, сайти-спільноти та ін.

Форуми — спеціалізовані сайти, на яких користувачі мережі Інтернет можуть безперешкодно обговорювати різні новини, проблеми, життєві сфери суспільства і т. п.

Форуми можуть бути як *спеціалізовані* (обговорюються одна або кілька проблем, пов'язаних між собою), так і *загальноспрямовані* (обговорюються проблеми, абсолютно різні за тематикою, часом зовсім не пов'язані одна з одною).

Соціальні мережі покликані надавати інтернет-користувачам можливість знаходити один одного і вести між собою спілкування в режимі онлайн. Кількість користувачів соціальних мереж невпинно зростає (рис. 11.12)

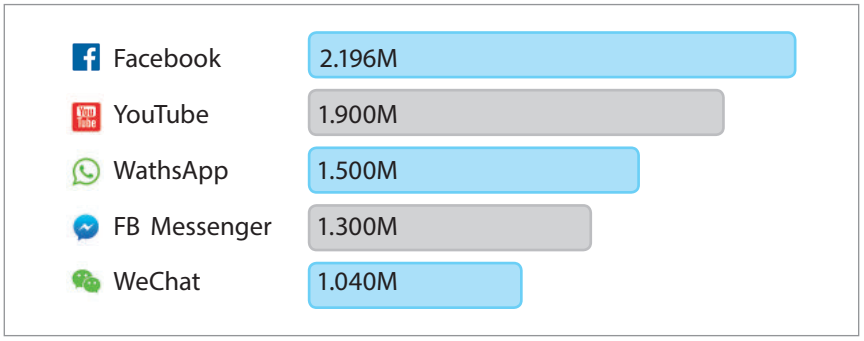


Рис. 11.12. Кількість користувачів соціальних мереж (у мільярдах)

Існують спеціалізовані соціальні мережі, які об'єднують людей, зі спільними інтересами (хобі, професія).

Соціальні мережі загальної спрямованості збирають в одне коло для спілкування людей, які зайняті в абсолютно різних сферах і мають різні інтереси і хобі (рис. 11.13).

Окрему категорію сайтів становлять освітні ресурси. У 9 класі ви ознайомилися з класифікацією освітніх та навчальних ресурсів. Пригадаємо, що освітніми інтернет-ресурсами називають ресурси освітнього характеру, розміщені у веб-просторі.

Розважальні портали впевнено тримають лідерство за популярністю нарівні з соціальними мережами. Найпопулярнішим є *відеохостинг*. Найбільш відвідуваними є ресурси, що безкоштовно пропонують відвідувачам онлайн перегляд і скачування фільмів, відеороликів. На другому місці — *сайти музичного напрямку*. За структурою вони подібні до відеохостингів, але пропонують безкоштовне прослуховування й скачування мелодій. Третє місце посідають класичні комплексні *сайти розваг*. Тут користувачі шукають інформацію відповідно до своїх інтересів: це може бути графіка, відео, тексти різної тематики і, безумовно, ігри.

Ігровий портал — складний розважальний інтерактивний проект, що передбачає велику відвідуваність і ресурсомісткість. На порталі зазвичай є розділ для пошуку потрібної інформації за темою, блок новин, і величезна кількість посилань на ігри з їхнім описом. Часто ігрові портали містять спеціалізовані й добре модеровані форуми.

Особливу увагу наразі приділяють відкритим онлайн-курсам на платформах **Prometheus**, **EdEra**, **EdX**, **Coursera**.

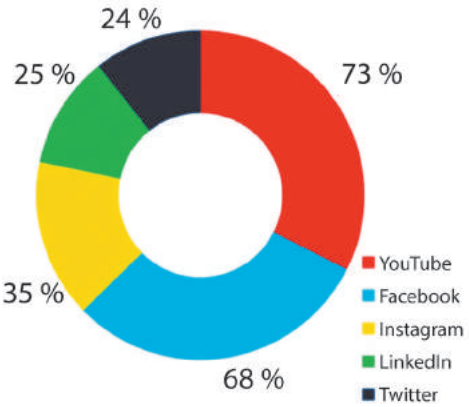


Рис. 11.13. Співвідношення користувачів найбільш впливових соціальних мереж (у відсотках)

Веб-сервіси нині отримали значне поширення (рис. 11.14).

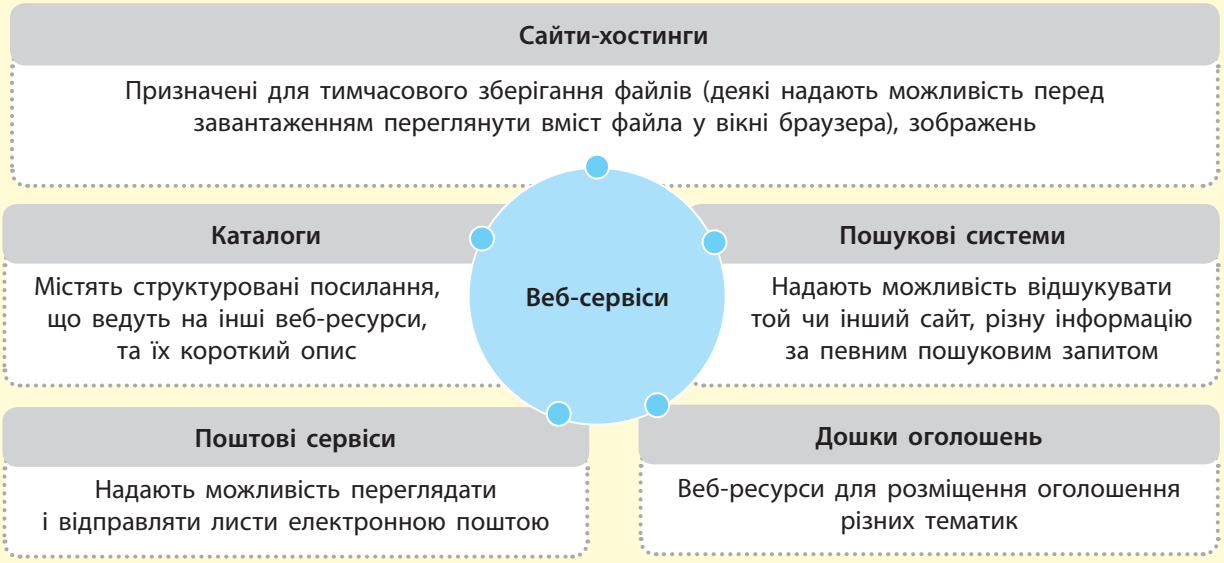


Рис. 11.14. Класифікація веб-сервісів

Сенс існування будь-якого сайта — це його відвідуваність. І якщо блогер публікує свою думку, розраховуючи на зворотний зв'язок у вигляді коментарів або лайків, то, наприклад, інтернет-магазин існує за рахунок реальних покупців, які шукають і, головне, купують товар, який їх зацікавив.

Цільова аудиторія сайта, цільові відвідувачі, — група інтернет-користувачів, на яку сфокусовано зміст сайта; коло відвідувачів, зацікавлених в інформації, товарах або послугах, що презентують на сайті.

Кожен сайт орієнтований на певну **цільову аудиторію**. Цільову аудиторію будь-якого сайта становлять відвідувачі, які зацікавлені:

- в отриманні інформації;
- у виборі товару або послуги;
- у придбанні товару чи послуги.

Цільові відвідувачі точно знають, в отриманні якої інформації вони зацікавлені і який саме товар або послугу бажають придбати.

Виділення цільової аудиторії з усіх відвідувачів сайта дозволяє точніше направити інформаційний або рекламний вплив і в результаті веде до розвитку бізнесу (збільшення продажів товарів або послуг).

Будь-який сайт, крім власне цільової аудиторії, має також побічну і випадкову аудиторію.

Побічною аудиторією є користувачі, які приходять з пошуку по запитах, суміжних із семантичним ядром сайта. Те саме можна сказати і про людей, які начебто «автоматично» клацають на рекламу, ще не знаючи, потрібна їм послуга чи ні. Вони можуть стати клієнтами, проте це відбувається нечасто. Отже, побічна аудиторія теж є цільовою.

Визначення цільової аудиторії полягає в складанні приблизного портрета цільового відвідувача сайта (так званого портрета клієнта).

Для отримання даних, які складають портрет цільового відвідувача, використовують різні способи збирання інформації.

Існує кілька способів збирання інформації про аудиторію сайта.

Спосіб 1	Лог-аналізатор сервера і дані лічильника відвідувань (дозволяють вивчити всі дії користувачів на сайті і конкретизувати розподіл аудиторії сайта за регіонами, за часом й ін.)
Спосіб 2	Опитування аудиторії сайта (анкетування унікальних відвідувачів з використанням опитувальної форми або реєстрації на сайті)
Спосіб 3	Опитування аудиторії на сайтах опитувань, поєднання панельних даних і даних лічильника відвідувань (анкетування відбувається на сайті панелі)
Спосіб 4	Системи аудиту і традиційні опитування дослідницьких компаній (агентств)
Спосіб 5	Моніторинг соціальних мереж

Дослідження великих аудиторій показують, що комунікацію легше налагоджувати з малими групами покупців, об'єднаними в одну цільову аудиторію.

Існує чотири основні принципи сегментації цільової аудиторії сайта (рис. 11.15).

Важливим джерелом статистичних даних про цільову аудиторію сайта є статистика запитів пошукових систем. Оцінити величину цільової аудиторії можна за кількістю пошукових запитів. Такий сервіс має Google (https//www.google.com/analytics).

Це безкоштовний сервіс, призначений для збору відомостей про відвідуваність, а також про дії інтернет-користувачів на сайті. Інструмент формує безліч звітів. Усебічний аналіз цільової аудиторії дозволяє оперативно реагувати на запити користувачів. Веб-майстер зможе дізнатися, як часто відвідувачі натискають на кнопки, на кшталт «оформити покупку» або «замовити зворотний дзвінок».

Google analytics — багатофункціональний сервіс для аналізу інтернет-сайтів і додатків.

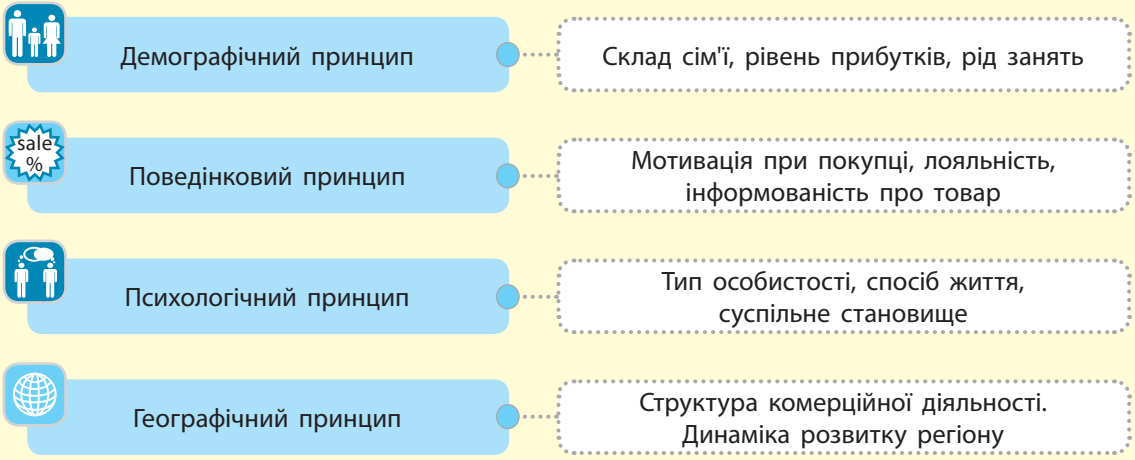


Рис. 11.15. Сегментація цільової аудиторії

У **Hootsuite** є багато можливостей для різноманітної аналітики. Наприклад, можна підключити **Google Analytics** зі свого сайта й дивитися графіки, порівнюючи з кількістю твітів і популярністю ваших посилань.

Socialmention шукає по більш ніж 100 соціальних медіа, включаючи мережі, соціальні закладки, блоги, форуми, соціальні сервіси та ін.

Socialbakers збирає статистику по сторінках інтернет-ресурсу. Власник сайта зможе дізнатися, як часто відвідувачі роблять переходи по внутрішніх і зовнішніх посиланнях.

Розробка українських програмістів — система моніторингу та аналізу посилань у соціальних медіа **YouScan** (https://youscan.io) — має потужний інструментарій аналітики і вміє розпізнавати зображення за допомогою штучного інтелекту. Серед клієнтів сервісу є такі брендові компанії, як McDonald`s, Ferrero, Bosch, Vodafone.

Групи в соціальних мережах допомагають не лише збільшити продажі компанії, а й просувати сайт компанії в пошукових системах. Існує безліч сервісів моніторингу авторитету сайта за кількістю згадок про нього в соціальних мережах, особливо в таких соціальних мережах, як Facebook, а також твітти із зазначеними посиланнями на сайт.

Hootsuite (http://hootsuite.com) — відмінний і, мабуть, один із найпопулярніших багатофункціональних сервісів для роботи з соціальними медіа. Акцент у сервісі зроблено на роботу з Twitter. У першу чергу Hootsuite буде корисний тим, хто веде кілька акаунтів відразу. Hootsuite успішно працює також з акаунтами Facebook, LinkedIn, MySpace і Foursquare, з блогами на WordPress; підключається до Ping.fm, що дозволяє оновлювати сторінки понад 40 соціальних мереж.

Socialmention (http://www.socialmention.com/) — одна з найпотужніших платформ для безкоштовного пошуку й аналізу інформації в соціальних мережах. Система шукає згадки у відібраних сервісах або в усіх відразу. Крім того, пропонує аналіз тональності згадувань, пов'язані ключові слова, популярні джерела та багато іншого. Дані можна експортувати або налаштувати їх отримання на e-mail.

Google Alerts (https://www.google.com/alerts) — простий і зрозумілий безкоштовний сервіс, який сканує публікації за заданими ключовими словами, висилаючи оповіщення по електронній пошті відповідно до частоти, налаштованої користувачем. Є можливість фільтрувати результати залежно від регіону, мови та джерел, у тому числі блогів і форумів.

Socialbakers (http://www.socialbakers.com) — масштабний статистичний сервіс, який називає себе «Серце статистики Facebook». Крім Facebook, Socialbakers безкоштовно надає моніторинг інформації за різними показниками в Twitter, Google+, LinkedIn тощо. Socialbakers відомий своїми рейтингами брендів на Facebook.

На основі даних, отриманих у результаті збору інформації, можна отримати зведені цифри, вивчити закономірності поведінки груп користувачів і оцінити ефективність рекламного впливу. Характеристики портрета цільової аудиторії враховуються при розробці дизайну і структури, при внесенні коректувань в інформаційне наповнення сайта з метою залучення більшої кількості відвідувачів, що входять у коло цільової аудиторії.

Запитання для перевірки знань

1. Скільки груп сайтів ви можете класифікувати?
2. Що входить до соціальних проектів?
3. Що таке цільова аудиторія сайта? Назвіть її основні характеристики.
4. Які основні групи цільової аудиторії сайта?
5. Назвіть найперспективніші, на вашу думку, способи збирання інформації про цільову аудиторію сайта.
6. Візуалізуйте будь-яким засобом результати вашого опитування, щодо того, які сайти є вподобаннями ваших друзів.

11.3. Інформаційна структура сайта

Будь-який сайт створюється для того, щоб його відвідували якомога більше користувачів. Як цього досягти?

Для користувача простота навігації сайтом — важливий чинник, що позитивно впливає на поведінкові фактори і, як наслідок, на видимість, позиції і трафік.

Інформаційна структура сайта — спосіб організації інформаційних даних на веб-сайті, а також структура взаємодії різних блоків інформації один з одним.

В ідеалі така структура повинна бути максимально зручною, щоб користувач міг швидко знайти потрібну йому інформацію. Інакше кажучи, структура сайта — це логічна схема побудови сторінок сайта з розподілом за папками і категоріями..

Сайти поділяються на сайти з *лінійною* та *ієрархічною* структурою (рис. 11.16). Також є сайти з *довільною* структурою, на яких навігація здійснюється в довільному порядку. Сайтом з довільною структурою є, наприклад, Вікіпедія. Структура сайта є дуже важливим фактором ранжування, покращуючи характеристики всього сайта.

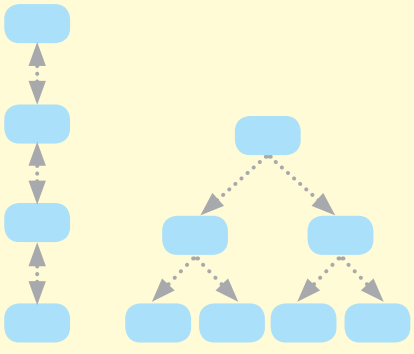


Рис. 11.16. Структура сайтів: лінійна (а) та ієрархічна (б)

Основною складовою будь-якого сайта є документ — це сторінка сайта, що має унікальну адресу в Інтернеті. Документи можуть бути як простими інформаційними сторінками, так і картками товару або, наприклад, лістингами товарів в інтернет-магазині.

Якщо сайт являє собою ієрархічну (деревоподібну) структуру, то головну сторінку можна уявити як «стовбур», а розділи і статті — як гілки й листя. Наскільки широкою буде структура, залежить від формату і типу сайта. Але навіть сайт, що складається з однієї сторінки, вже є основою, або «стовбуром», такої структури, від якої він може розвиватися в різних напрямках.

Рис. 11.17. Структура сайта-візитки

Ієрархічна структура є базисом для будь-яких інших видів структури, в яких змінюється вид зв'язку документів — чи це лінійний зв'язок, де кожен документ пов'язаний з наступним і з головною, чи більш широкий, де кожен документ пов'язаний між собою і з головною.

Принцип побудови зв'язків між документами залежить від типу сайта. Загалом можна виділити такі типи сайтів, які мають відмінності в структурах: сайт-візитка (рис. 11.17), комерційний сайт (рис. 11.18), інтернет-магазин, інформаційний сайт. Кожен тип сайта передбачає свій вид структури.

Наразі все більшу популярність набувають так звані *посадкові*, або *лендінг* (від англ. *landing page*), сторінки. Посадкова сторінка розробляється з урахуванням психології відвідувачів, вона має чіпляти й не відпускати їх до самого моменту покупки (або реєстрації). Це може бути передплатна сторінка, завдання якої зводиться до отримання електронної адреси користувача, який зайшов на сторінку. Так збирається база.

Товарна сторінка служить для продажу товару. Це просунута картка, яку можна зустріти в будь-якому інтернет-магазині, з описом товару, умовами доставки, контактним телефоном. Головною тут буде кнопка **Купити**. Аналогічно працює лендінг з послуг, коли намагаються продати послугу. Є контактний телефон, за яким можна цю послугу замовити.

Структура необхідна для індексації сайта пошуковими системами. У пошукових систем є низка своїх вимог до структури. Чим більш правильно й логічно вона спроектована, тим простіше пошуковій системі проіндексувати сторінки і надати їх користувачеві.

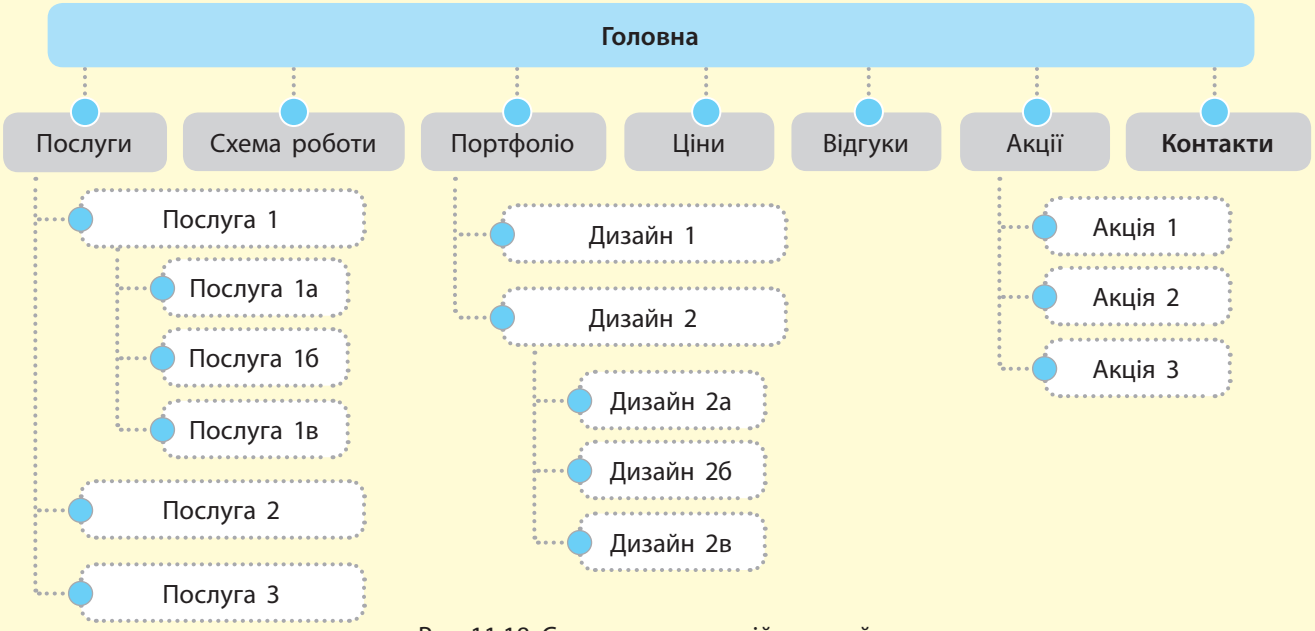


Рис. 11.18. Структура комерційного сайта

Правильно розроблена структура сайта дає можливість пошуковим системам швидко здійснити обхід ресурсу, а відвідувачам легко переміщуватися між веб-сторінками.

Складність структури сайта визначається двома параметрами: рівнем вкладеності і збалансованістю.

Рівень вкладеності — це кількість переходів, які потрібно зробити з головної сторінки до найдальшого документа в структурі. Не рекомендується, щоб це значення було більше від 3. *Збалансованість* визначають «на око», оцінюючи кількість документів усередині розділів і рубрик. Не повинно бути помітних розбіжностей, коли в одному розділі вміщено 100 документів, а в іншому — 25.

Для побудови структури сайта в цілому підійде будь-яка програма, яка може працювати з діаграмами. Створення схеми дозволяє візуально оцінити складність структури й охопити всі її напрямки поглядом.

Найбільш зручним для створення структур сайта є таке програмне забезпечення.

- X-mind — безкоштовна програма для побудови інтелект-карт, структур і діаграм різних процесів. Вона досить проста у використанні, має широкий функціонал і підтримується всіма операційними системами, використовується на мобільних пристроях (рис. 11.19).
- Draw.io — безкоштовний онлайн-сервіс для створення структур і діаграм. Він вимагає наявності пошти від Google. В арсеналі має більше інструментарію, ніж у X-mind, наявна низка технічних і інженерних шаблонів.

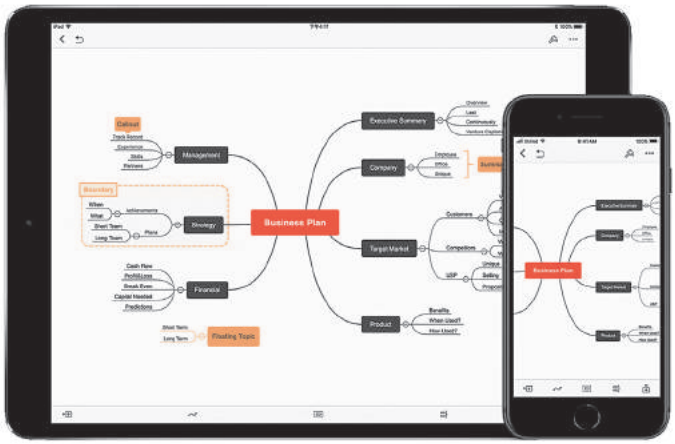


Рис. 11.19. Використання **X-mind** на мобільних пристроях

Ще один плюс **Draw.io** — можливість завантаження результату на Google-диск та інші сервіси нарівні із завантаженими на комп'ютер.

На рис. 11.20 подано структуру сайта вивчення інформатики, розроблену за допомогою Draw.io.

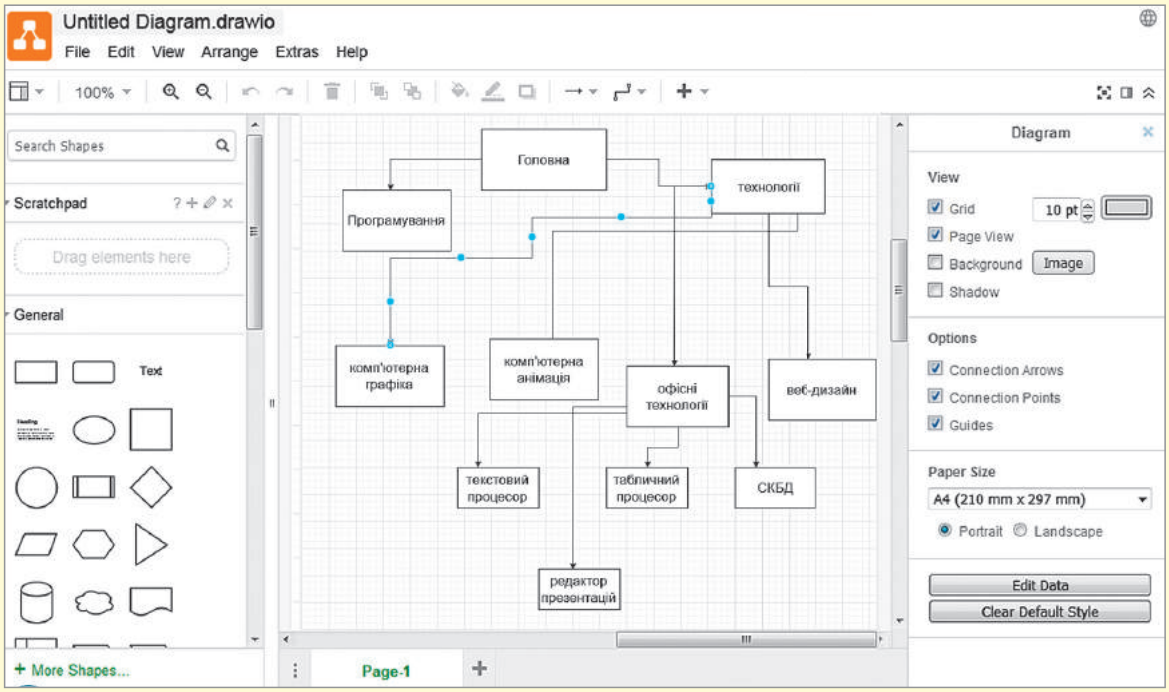


Рис. 11.20. Структура сайта вивчення інформатики, зроблена за допомогою **Draw.io**

Запитання для перевірки знань

1. Що таке інформаційна структура сайта?
2. Назвіть параметри складності структури. Як вони розрізняються?
3. Поміркуйте, яка структура характерна для будь-якого сайта. Опишіть цю структуру.
4. Яка мета створення посадкової сторінки?
5. Обґрунтуйте, чим важлива структура сайта.
6. Створіть за допомогою будь-якого онлайн-сервісу структуру інформаційного сайта; інтернет-магазину.

11.4. Системи керування вмістом

Уявіть, що вам потрібно створити власний сайт. Із чого потрібно розпочати?

Для створення власного сайта спочатку необхідно розробити макет, створити його за допомогою тегів HTML, селекторів CSS і сценаріїв JS. Потім розмістити сайт на певному сервері, наповнити його контентом і постійно оновлювати. При цьому сайт бажано зв'язати із соціальними мережами, надати можливість зворотного зв'язку.

Це означає, що потрібно знати й уміти використовувати різноманітні API, програмувати мовою PHP і використовувати безліч інших технологій. Тобто ми одночасно маємо бути front-end- і back-end-розробниками.

Front-end — інтерфейс взаємодії між користувачем та базовою апаратно-адміністративною частиною (back-end). Інакше кажучи, front-end — це користувацький інтерфейс, а back-end — адміністративний, пов'язаний із керуванням сайтом.

API (англ. *Application Programming Interface*) — інтерфейс програмування застосунків, що створює можливість використовувати ресурси інших програм.

PHP (англ. *Hypertext Preprocessor* — гіпертекстовий процесор) — мова програмування, створена для генерації HTML-сторінок на веб-сторінці сайта.

Людина, якій потрібен сайт, має або оволодіти досить потужним інструментарієм, або звернутися до професіоналів. Ще один вихід — це знайти такі інструменти, які б допомогли непрофесіоналам створювати та супроводжувати свої сайти, інтернет-магазини, блоги тощо.

На першому етапі появи сайтів CMS-рушіїв ще не було, тому більшість сторінок були статичними. Шаблони використовувалися тільки в складних настільних **WYSIWYG**-програмах редагування HTML-тексту або додавалися вручну в текстовому редакторі. Навігація будувалася вручну з використанням фреймів або на основі клієнтського javascript-програмування. Такі сайти важко адмініструвалися, причому вручну через ftp-протоколи. Першою CMS вважають **Vignette**, представлену в листопаді 1995 року. Вона дозволила користувачам-аматорам створювати, редагувати та відстежувати вміст сайта та публікувати його в мережі.

Система керування вмістом (англ. Content Management System, або CMS) — програмне забезпечення для організації спільного процесу створення, редагування й керування контентом веб-сайтів.

Принцип роботи всіх CMS заснований на поділі контенту (вмісту) і дизайну (оформлення) сайта.

Ознайомимося з основними функціями CMS (рис. 11.21).

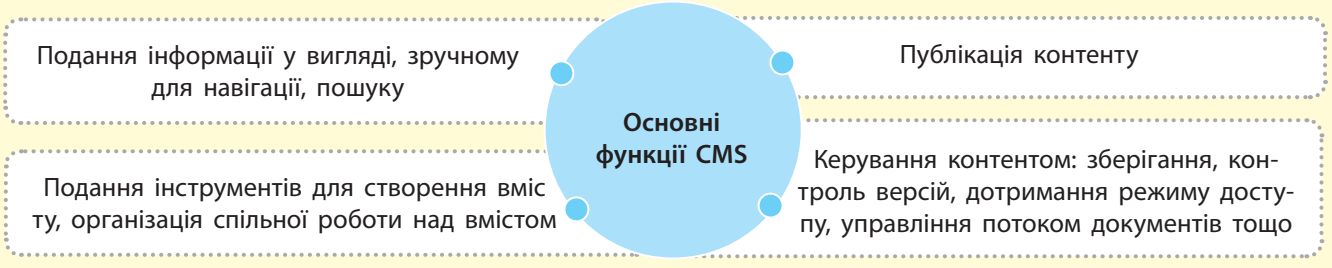


Рис. 11.21. Основні функції CMS

Дизайн сайта зазвичай змінюється рідко. Натомість зміна контенту сайта може відбуватися не лише щодня, а й щогодини. Тому CMS споряджені так званими шаблонами — спеціальними «порожніми» заготовками сторінок, у яких прописано дизайн сайта, залишилося лише наповнити їх інформацією.

Кожна CMS має у своєму складі WYSIWYG-редактор (акронім від англ. *What You See Is What You Get* — що бачиш, те й отримуєш). Сторінку одного з них наведено на рис. 11.22.

За зовнішнім виглядом WYSIWYG дуже схожий на звичні текстові редактори офісних застосунків, тому користувачеві не надто складно освоїти його. А наявність у системі великої кількості готових шаблонів дає змогу обрати потрібний дизайн буквально за лічені хвилини.

Magento — одна з найпопулярніших CMS у сегменті **OpenSource** рішень для організації електронної комерції в мережі (створення інтернет-магазинів).

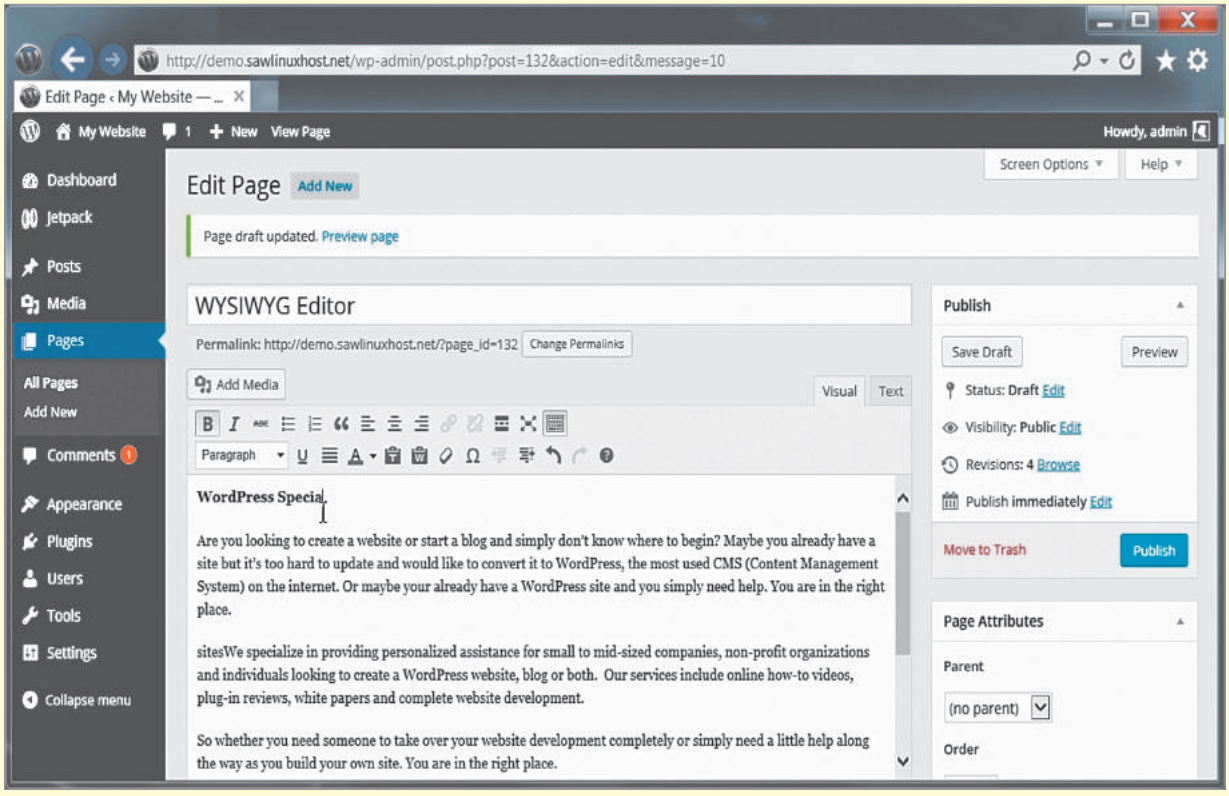


Рис. 11.22. Сторінка WYSIWYG-редактора CMS **WordPress**

У процесі створення сайта за допомогою такого редактора сторінка відразу відображається в такому вигляді, ніби переглядається браузером (на відміну від звичайних редакторів, де записується послідовність команд-тегів).

Для відображення такої сторінки необхідно викликати браузер. При цьому теги стають невидимими, і сторінка матиме вигляд, відповідний встановленій тегами розмітці.

Інтерфейси візуального і текстового редактора схожі, тому користувачеві легко його опанувати. Наявність великої кількості готових шаблонів дає змогу обрати потрібний дизайн за лічені хвилини.

Існує понад 240 тисяч інтернет-магазинів, створених на базі платформи **Magento Commerce**. Із 2010 до 2015 року була у складі корпорації **eBay**. Платформа **Magento Commerce** становить близько 30 % загальної частки ринку.

Ознайомимося з найбільш популярними безкоштовними CMS (рис. 11.23).

	WordPress — найкращий вибір для початківців завдяки простоті використання: постійно вдосконалюється. Система особливо зручна для невеликих та середніх стиль-сайтів, блогів і невеликих інтернет-магазинів.
Joomla!	Joomla проста у використанні, проте для новачка буде складнішою від WordPress, оскільки для будь-якої зміни дизайну потребує навичок програмування. Система є найліпшою для електронної комерції або сайтів соціальних мереж.
Drupal	Drupal орієнтована насамперед на досвідчених програмістів, має потужний інструментарій із широкими можливостями. Система ідеальна для великих і серйозних сайтів, для яких необхідні знання програмування. Drupal потребує глибоких знань HTML, CSS і PHP.

Рис. 11.23. Порівняльні характеристики найпопулярніших CMS

Приклади стартових сторінок CMS наведено на рис. 11.24.

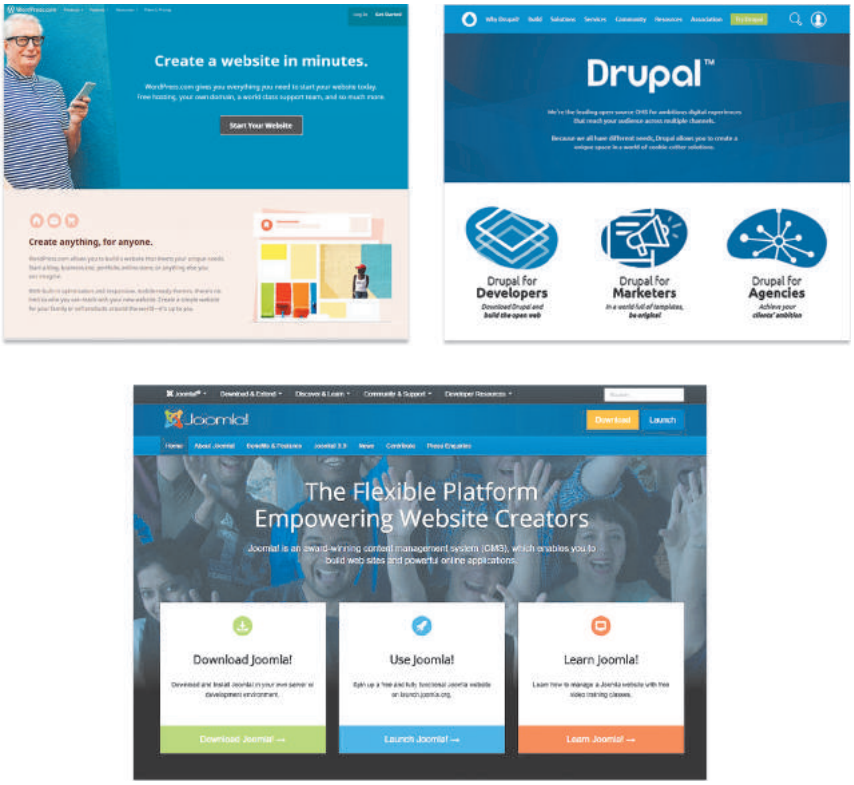


Рис. 11.24. Приклад стартових сторінок CMS

Запитання для перевірки знань

1. Чим відрізняється front-end від back-end?
2. Що таке CMS?
3. Назвіть функції систем керування вмістом.
4. Що таке шаблон сторінки?
5. Що таке WYSIWYG? Наведіть приклади безкоштовних візуальних редакторів.
6. Дослідіть CMS і зробіть висновки. Скористайтеся посиланням websitesetup.org/cmscomparison-wordpress-vs-joomla-drupal

Завдання для самостійного виконання

У 9 класі ми створювали сайт за допомогою **Google Sites**. Поміркуйте, чи можна вважати цей інструмент CMS. Обгрунтуйте свою відповідь.

11.5. Адміністрування сайта

Поміркуйте, чи можете ви змінювати інформацію в будь-якому сайті, який ви відкриваєте браузером?

Недостатньо просто створити сайт. Необхідно робити постійні оновлення сайта, поновлювати його контент.

Адміністрування сайта — це комплекс заходів щодо підтримки чіткого функціонування сайта, його працездатності, швидкої роботи, зручності для користувача, регулярного розміщення матеріалів на його сторінках.

Тобто адміністрування веб-сайту — це безперервна підтримка його працездатності, що додатково полягає у виконанні на сайті супровідних робіт: публікації матеріалів, додаванні форм, віджетів, банерів, редагуванні зображень.

Необхідно відрізняти відвідувача від адміністратора сайта. На відміну від пересічного відвідувача, який може переглядати сайт, заповнювати форму, додавати коментар, адміністратор сайта на будь-якій сторінці сайтів, які він супроводжує (адмініструє), у куточку бачить елемент, який вмикає режим редагування. При редагуванні зверху з'являється панель інструментів (toolbar) із вибором стилів тексту, налаштування зображення або керування таблицею залежно від редагованого об'єкта. Крім того, обов'язково має бути кнопка Зберегти зміни. Така панель називається панеллю адміністрування.

Панель адміністрування — це панель, яка містить посилання на екран адміністрування, такі як додавання нової публікації, перегляд коментарів, редагування профілю тощо.

Функції адміністративної панелі такі: створювати і редагувати сторінки, змінювати меню та інші елементи сайта.

Приклад такої панелі наведено на рис. 11.25.

Адміністратори веб-сайтів, також відомі як веб-майстри (webmaster), або адміністратори мережевих і комп'ютерних систем, несуть відповідальність за всі аспекти збереження вмісту та дизайну веб-сайту. Зазвичай вони тісно співпрацюють з клієнтами, щоб переконатися, що розуміють їхні побажання щодо вигляду та роботи веб-сайтів. Вони також можуть відповідати за належне функціонування локальних мереж клієнтів.

Рис. 11.25. Приклад адміністративної панелі CMS **Joomla**

Адмін-панель, шаблон панелі адміністрування — готовий набір для створення панелі адміністрування сайтом. Продукт, готовий для використання, в який входять необхідні компоненти, віджети, елементи інтерфейсу.

Адмін-шаблони різняться базовим фреймворком, на якому вони розроблені, дизайном, набором версій і модифікацій під відомі платформи розробки.

Часто робота з адміністрування полягає в обробці надісланих клієнтом текстів, вони повинні стати конкретними, точними, не містити граматичних і пунктуаційних помилок тощо. Ключовим елементом адміністрування є регулярна технічна підтримка, яка дозволить убезпечити ресурс від можливих технічних неполадок і збоїв. Сюди також відносять регулярну перевірку актуальності використовуваного доменного імені, вирішення маловідомих технічних питань, впоратися з якими можуть тільки фахівці.

Адміністрування сайта умовно можна розподілити на інформаційне та технічне. Розглянемо таблицю.

Таблиця. Види адміністрування сайта

Вид	Опис
Інформаційне	Регулярне додавання новин, статей, товарів, оголошень та інших матеріалів залежно від тематики сайта: • робота з наповнення сайта матеріалами; • редагування матеріалів і виправлення помилок; • оптимізація й підвищення зручності використання розміщеного контенту; • підтримка актуальності та практичної цінності матеріалів; • своєчасне вилучення й архівація застарілої інформації
Технічне	Забезпечення постійного стабільного доступу до сайта, вибір і налаштування хостингу, цілодобовий контроль стану сервера, підтримка сайта в цілодобовому робочому стані

Розглянемо елементи інформаційного адміністрування.

- Робота над створенням зрозумілої та зручної структури сайта (без застосування мов програмування), формування швидкої навігації. Сюди входить створення нових розділів, пунктів меню, сторінок тощо.
- Наповнення сайта. Для цього використовується статейний і новинний матеріал, прайс-листи, різноманітні медіаматеріали, карти та контактні дані. Матеріали мають відповідати дійсності і постійно оновлюватися. Інакше кажучи, необхідна постійна робота з контентом. Контент — це вміст сайта. Це текст, зображення, аудіо- та відеоматеріали. Контент повинен бути унікальним, інформативним, привабливим. Інформаційне адміністрування передбачає роботи не лише зі створення контенту, а й редагування.
- Робота з опрацювання надісланих клієнтом текстів, які повинні стати точними й грамотними.

Найважливіший аспект роботи веб-адміністратора — це захист сайта від вірусів і шкідливих програм. Адміністратор веб-сайту повинен знати про можливі атаки ботів і шкідливих програм і мати необхідні для боротьби інструменти. Неубезпечений веб-сервер може стати об'єктом для хакерів, які будуть використовувати сервер або безпосередньо для атаки, або для перетворення його на зомбі, який, наприклад, щосекунди розсилає спам-повідомлення.

Технічна підтримка необхідна будь-якому сайту, вона передбачає низку заходів, спрямованих на підтримку безперебійної роботи сайта.

Основні функції технічного адміністрування:

- забезпечення безперебійної та швидкої роботи сайта;
- відстеження й усунення різних збоїв і вірусів;

- функціональна підтримка сайта: встановлення й оновлення модулів (інтеграція з платіжними системами, форм тощо), налаштування корпоративної пошти.

Технічне адміністрування передбачає вибір хостингу, його налаштування, підготовку сайта до переносу, резервне копіювання всієї важливої інформації, роботу з базами даних. Така робота може передбачати повноцінне наповнення товарами інтернет-магазинів, включення в діяльність seo-інструментів (див. § 11.23), які спрямовані на поліпшення позицій сайта в пошукових системах, інколи адміністратор може відповідати за створення та управління інтернет-рекламою сайта.

За потреби адміністратор має створювати поштові скриньки сайта, відповідати на запитання відвідувачів, керувати доступом до редагування сайта.

Наведемо приблизний перелік обов'язків адміністратора сайта.

Для ефективної роботи сайта необхідно забезпечити цілодобовий і безперебійний доступ до нього. Таку можливість створюють хостинг-провайдери (див. § 3.17), які надають в оренду певний дисковий простір на своїх серверах.

- Для дотримання відповідності контенту сайта певному напрямку потрібно стежити за інформаційним наповненням сайта, формувати авторський колектив і добирати тематику текстів.
- Адміністратор зобов'язаний стежити за інформацією, що надходить на сайт, за необхідності редагуючи та коментуючи її.
- Він відстежує становище свого інтернет-ресурсу серед конкурентів і на мережевому ринку, працює над створенням методів залучення більшої кількості відвідувачів. Треба вивчати інтереси й запити інтернет-користувачів.
- Адміністратор повинен стежити за дотриманням правил усіма відвідувачами, убезпечувати сайт від можливих технічних несправностей і хакерських атак.
- Йому необхідно вміти працювати з системами управління контентом, розуміти, як з технічної точки зору функціонує та чи інша частина сайта і що необхідно зробити, якщо одна з них перестане працювати належним чином. Для цього адміністратору стане в пригоді знання HTML, CSS, PHP, JavaScript, Adobe Flash, а також інших більш складних мов і програм.
- Адміністратор повинен піклуватися про захист інформації та персональних даних, що потрапляють на сайт. Часом адміністратору доводиться займатися додатковим супроводом сайта, наприклад оптимізувати його для пошукових систем.
- Перебої в роботі сайта зазвичай виникають раптово, а налагодити його роботу потрібно в найкоротші терміни. Тож адміністратор, з одного боку, повинен вміти працювати в стресовій ситуації та залишатися розсудливим, незважаючи ні на що, а з іншого — мати аналітичний склад розуму, щоб оперативно виявити проблему та знайти шляхи її розв'язування.

Запитання для перевірки знань

1. Що таке адміністрування сайта?
2. На які види поділяється робота з адміністрування сайта?
3. Що входить в інформаційне адміністрування?
4. Які послуги надаються в плані технічного адміністрування?
5. Знайдіть в Інтернеті відомості про компанії, які надають послуги адміністрування. Створіть інфографіку їх послуг.
6. Якби вам запропонували бути адміністратором сайта вашого навчального закладу, щоб ви запланували зробити в першу чергу?

11.6. Інструменти веб-розробника

Які вміння й навички будуть корисними під час розробки сайта?

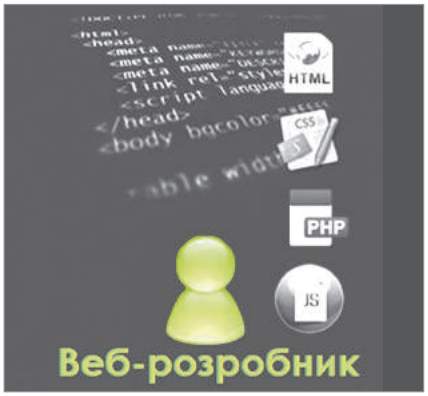


Рис. 11.26. Інструменти веб-розробника

Є цілий клас завдань (створення сайта, перевірка його працездатності, супровід, оптимізація та просування), виконання яких неможливе без участі фахівців різної спеціалізації.

Який вигляд матиме сайт — цим питанням опікується веб-дизайнер. Веб-розробник отримує результат його роботи у вигляді макету. Його завдання — написати код, який відобразить браузер у саме такому вигляді, як запропонував веб-дизайнер (рис. 11.26).

Усі веб-сайти будуються з допомогою мови розмітки HTML та каскадних таблиць стилів CSS.

Редактор коду — це перший інструмент веб-розробника. Сучасні редактори надають широкий асортимент інструментів, які полегшують і прискорюють процес розробки коду.

Розглянемо найбільш поширені нині редактори коду згідно зі статистикою сайта Stack Overflow (рис. 11.27).

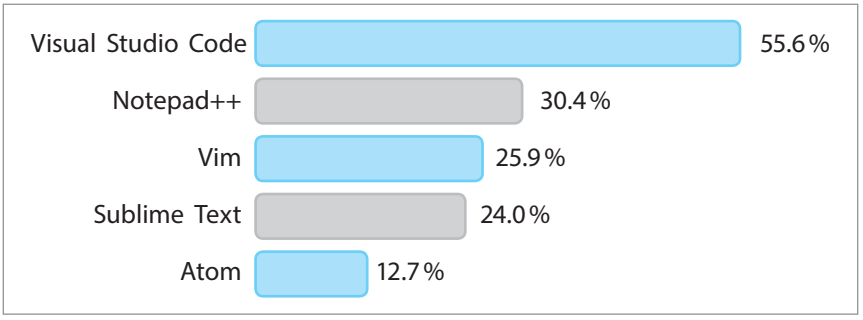


Рис. 11.27. Рейтинг популярності редакторів коду

Visual Studio Code — це безкоштовний крос-платформний редактор коду, розроблений корпорацією Microsoft (рис. 11.28). Виходячи з опитування, проведеного Stack Overflow у 2019 році, Visual Studio Code є одним із найпопулярніших редакторів коду, ним користуються понад 55 % розробників.

Програма має відкритий вихідний код. Вона оснащена доступним набором інструментів для редагування й налаштування, легко інтегрується з іншими сервісами, її властивості можна легко розширити.

Notepad++ — це розвинутий редактор коду, випущений у 2003 році і доступний лише на платформі Windows (рис. 11.29). Виходячи з опитування, проведеного Stack Overflow у 2019 році, Notepad++ займає другу позицію в рейтингу популярних редакторів коду. Ним користуються понад 30 % розробників.

Програма дуже швидка, підтримує зовнішні плагіни, включаючи макроси, її інтерфейс передбачає редагування в різних вкладках, є опція перетягування для початківців. Notepad++ підтримує повноекранний режим, робить

stackoverflow

Сайт **Stack Overflow** створено у 2008 році Джефом Етвудом і Джоелом Спольські як ресурс для програмістів, які намагаються розв'язати якусь проблему. Це сервіс питань і відповідей з програмування, який входить до сотні найбільш відвідуваних сайтів. Щомісяця — близько 50 млн людей, 72 % із яких є веб-розробниками. Нинішні й майбутні фахівці обговорюють сотні мов програмування, фреймворків, платформ тощо. Причому якість відповідей на **Stack Overflow** тримається на рівні кращих академічних стандартів. Тому компанії, запрошуючи на роботу розробника ПЗ, розглядають активний профіль на **Stack Overflow** як важливу частину резюме.

автоматичні відступи та автодоповнення, має дуже продумане підсвічування синтаксису. Програма дозволяє здійснювати пошук і заміну тексту, перевірку правопису з порівнянням файла, використовувати фолдинг (функціонал згортання).

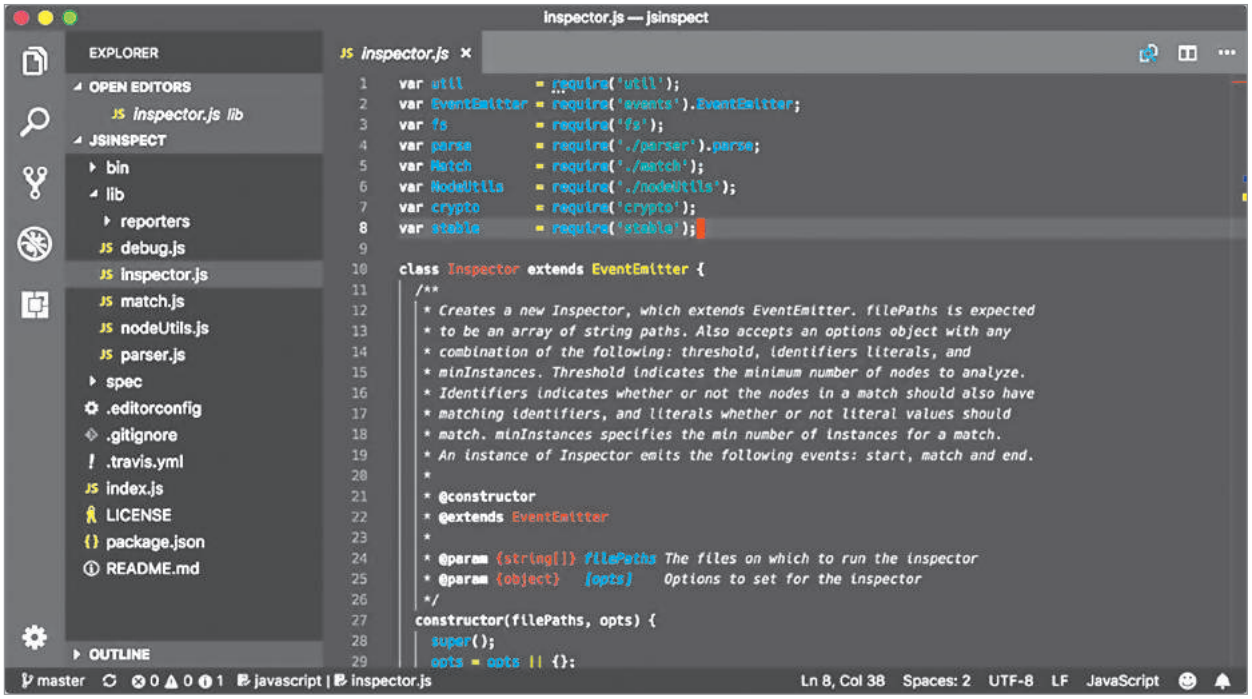


Рис. 11.28. Приклад робочого вікна **Visual Studio Code**

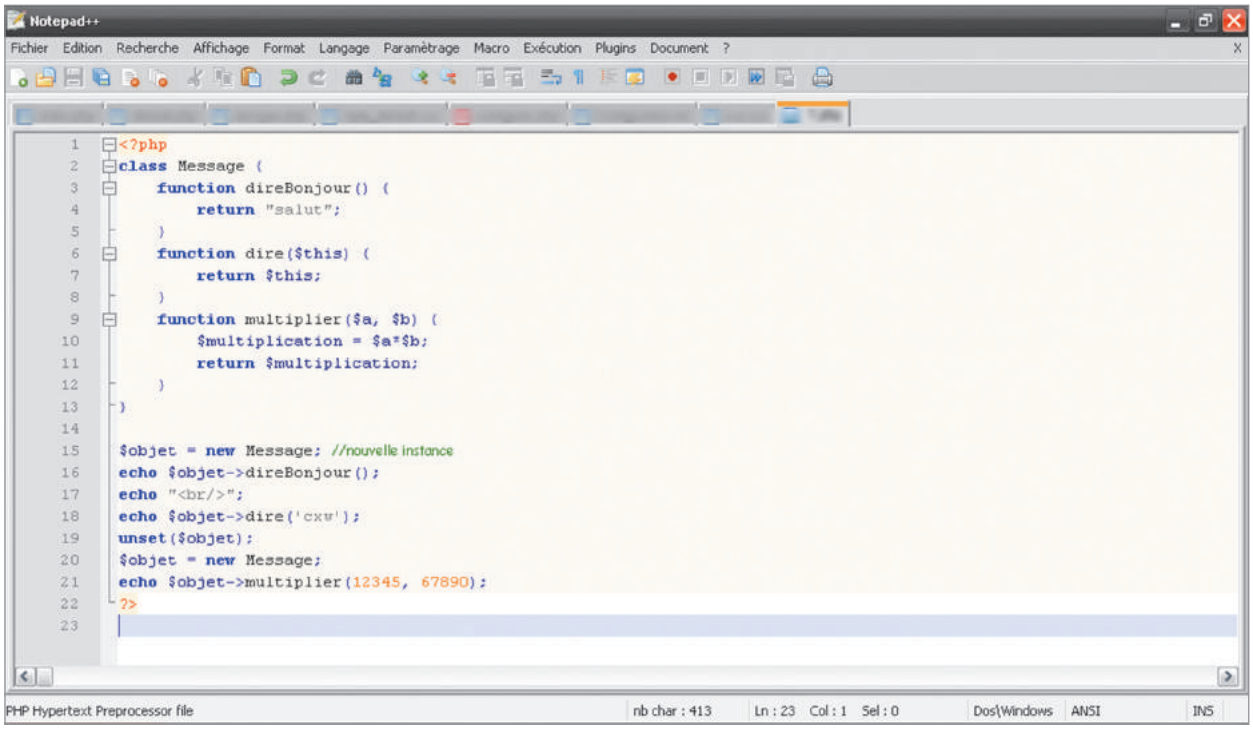


Рис. 11.29. Приклад робочого вікна **Notepad ++**

Sublime Text 3 — це крос-платформний редактор коду, що має як преміум-версію, так і безкоштовну версію (рис. 11.30), яку можна завантажити на офіційному сайті. Існує багато редакторів коду, які підтримують чорне тло для розширеного перегляду. Sublime Text 3 — один з них.

Програма легка й дуже швидка в роботі, за замовчуванням надає автодоповнення, підсвічування синтаксису та фолдинг, має зручний інтерфейс для початківців, виявлення та виділення синтаксичних помилок, дозволяє одночасно редагувати багато рядків. У Sublime Text 3 можна додатково налаштувати плагін Package Control, додавши таким чином інструменти налаштування, нові теми.

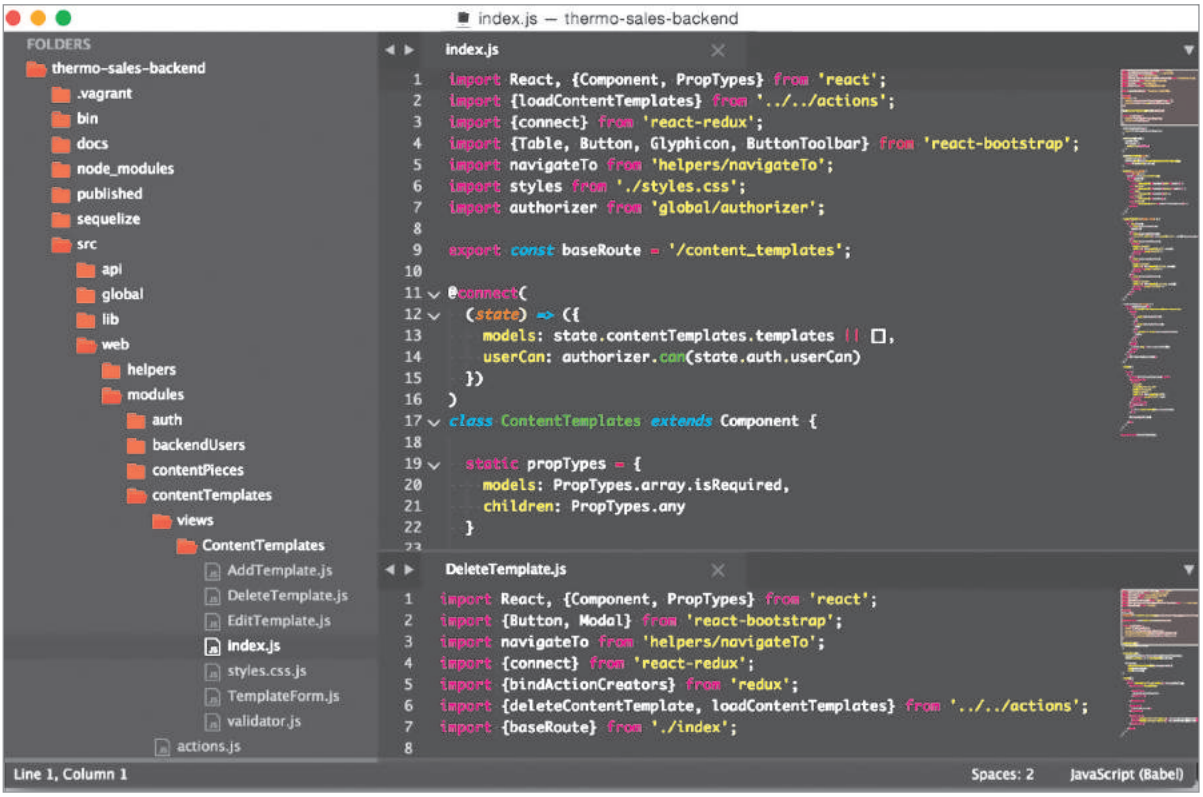


Рис. 11. 30. Приклад робочого вікна **Sublime Text 3**

Atom — це безкоштовний крос-платформний редактор із відкритим вихідним кодом (рис. 11.31). Це розробка GitHub. Інтерфейс Atom виглядає як клон редактора Sublime Text, проте працювати в ньому набагато комфортніше, оскільки він більш простий і зрозумілий. Крім того, у випадку коли розробник під час програмування стикається з труднощами, спільнота GitHub досить активно розв'язує цю проблему.

Програма за замовчуванням надає підсвічування синтаксису, доповнення й згортання коду, а також має вбудовану підтримку десятків мов програмування. Atom постачається із вбудованим менеджером пакетів, завдяки чому можна здійснювати пошук, установлювати або створювати власні пакети для розширення функціоналу редактора.

Подібно **Visual Studio Code** **Atom** оснащений потужним інструментом для парного програмування — **Teletype**. Це дає можливість кільком розробникам приєднуватися до ізольованої сесії й працювати спільно.

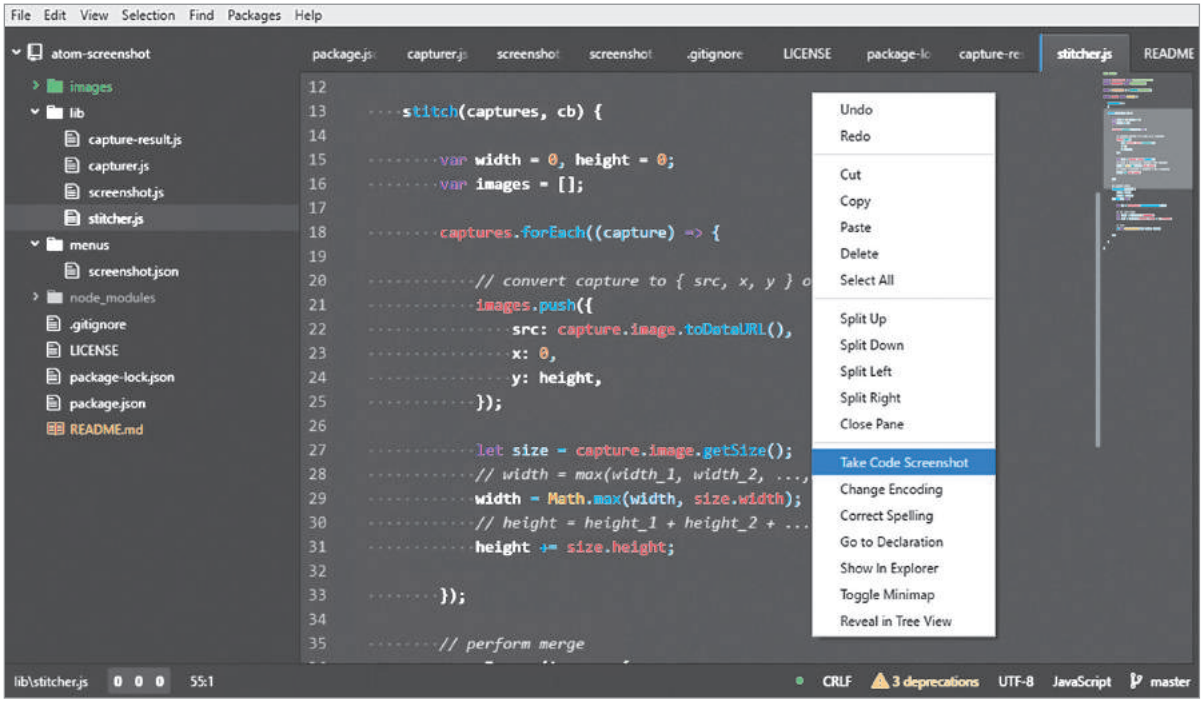


Рис. 11.31. Приклад робочого вікна **Atom**

Vim — це редактор (від англ. *vi improved* — покращений *vi*), який уперше був випущений наприкінці 1991 року Брамом Мооленааром (рис. 11.32). Справжню міць Vim демонструє під час роботи з структурованими текстами та є незамінним для програмістів і верстальників, його використовує кожен четвертий веб-девелопер.

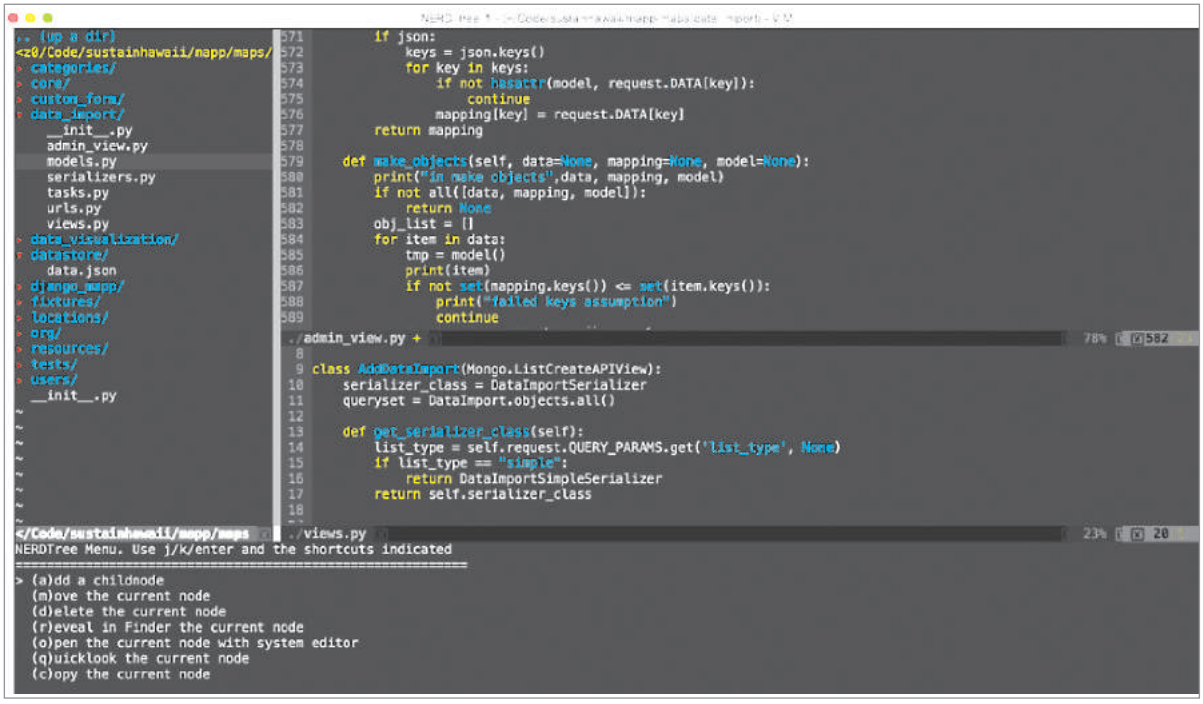


Рис. 11.32. Приклад робочого вікна **Vim**

Vim є клоном текстового редактора vi для Unix, який написав Біл Джой у 1976 році. Тоді ж був придуманий його незвичайний інтерфейс, заснований на поділі режимів роботи: стандартний режим, режим вставки, режим командного рядка.

Brackets — це редактор коду, створений з нуля спеціально для веб-дизайнерів і фронтенд-розробників, які працюють переважно з JavaScript, HTML і CSS (рис. 11.33).

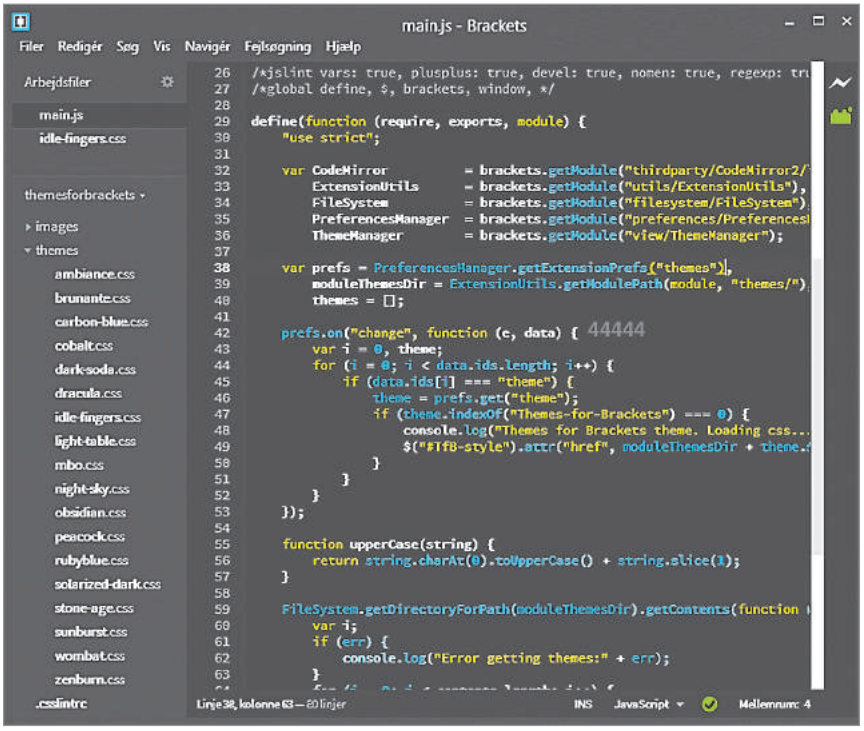


Рис. 11.33. Приклад робочого вікна **Brackets**

Програма поставляється з основними стандартними властивостями, включаючи автодоповнення, підсвічування синтаксису для багатьох мов програмування, підтримку швидкого редагування й різноманітних препроцесорів.

У Brackets є властивість extract, що дозволяє підтягувати кольори, розміри, градієнти, шрифти та інші важливі дані з PSD-файла в CSS-файл, готовий до використання. Brackets дуже популярний серед українських веб-розробників.

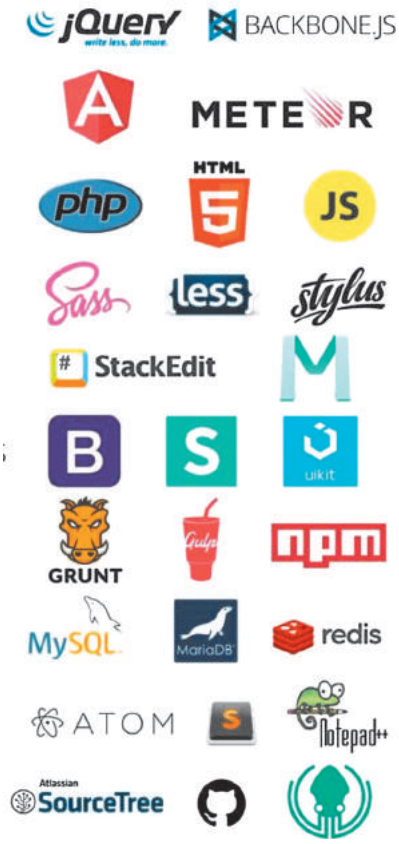


Рис. 11.34. До завдання 6

Запитання для перевірки знань

1. У чому полягає різниця між роботою веб-дизайнера і веб-розробника?
2. Якими інструментами користується в роботі веб-розробник?
3. Назвіть найпопулярніший, на вашу думку, редактор коду.
4. Яке призначення редактора коду?
5. Що об'єднує всі редактори коду?
6. На рис. 11.34 наведено логотипи інструментів веб-розробника. Знайдіть в Інтернеті відомості про ці інструменти. Створіть класифікацію інструментів.

11.7. Мова гіпертекстової розмітки

З чим стикається користувач будь-якої CMS?

Для розуміння процесу створення сайта необхідне опанування мови розмітки гіпертексту.

HTML (англ. *HyperText Markup Language*) — стандартна мова розмітки документів у Всесвітній павутині, яка обробляється спеціальними програмами (браузерами) і відображається у вигляді документа у зручній для людини формі.

Слід зазначити, що HTML не є мовою програмування, вона призначена лише для розмітки сторінки, надання певного вигляду її складовим.

Гіпертекст — електронний документ, який містить зв'язки з іншими електронними документами. Такі зв'язки називаються гіперпосиланнями.

HTML створювалася як мова для обміну науковою й технічною документацією, як один із компонентів технології розробки розподіленої гіпертекстової системи World Wide Web (яку ми звикли називати Всесвітньою павутиною).

Ідея така: користувач має можливість переглядати документи (сторінки тексту) у найзручнішому для себе порядку, а не послідовно, як це узвичаєно під час читання книжок. Це досягається створенням спеціального механізму зв'язування різних сторінок тексту за допомогою гіпертекстових посилань.

Будь-який документ мовою HTML є набором елементів, водночас початок і кінець кожного елемента позначається спеціальними позначками — тегами.

Теги — команди мови HTML. HTML-теги — це ключові слова або символи, які записуються в кутових дужках.

Теги бувають двох видів: парні й непарні (їх ще називають поодинокими). Парні теги складаються з відкриваючого і закриваючого тегів. Теги нечутливі до регістра (тобто регістронезалежні), тому можуть бути написані як великими, так і малими літерами.

Теги визначають, де починається й де закінчується HTML-елемент.

Текстові документи, що містять розмітку мовою HTML (такі документи зазвичай мають розширення .html або .htm), опрацьовуються спеціальними застосунками, які відображають документ у його відформатованому вигляді. Такі застосунки називаються браузерами.

Пригадаємо, що *браузером* (або *веб-переглядачем*) називають програмне забезпечення для комп'ютера або іншого

Мову HTML розробив британський учений Тім Бернерс-Лі приблизно в 1989–1991 роках під час роботи в Європейській лабораторії з ядерних досліджень ЦЕРН у Женеві (Швейцарія).

Нині найпопулярнішими браузерами є **Mozilla Firefox**, **AppleSafari**, **Google Chrome**, **Opera**, **Internet Explorer** (**Microsoft Edge**).

Для тих, хто хоче дізнатися більше про створення Всесвітньої павутини, радимо переглянути фільм BBC The Virtual Revolution /The Great Levelling (https://www.dailymotion.com/video/x5vfk2c)

Контент (англ. *content* — вміст) — це інформаційне наповнення сайта.
Контент може містити:
- Текст
- Статичну графіку (§ 11.12)
- Анімаційні елементи (§ 11.13)
- Мультимедійні елементи: відео, аудіо (§ 11.14)

електронного пристрою, під'єднаного до Інтернету, який дає змогу користувачеві взаємодіяти з текстом, малюнками або іншою інформацією на гіпертекстовій веб-сторінці.

За допомогою тегів браузер розпізнає структуру документа. Отримавши цю інформацію, браузер використовує вбудовані в нього за замовчуванням правила про те, як відображати контент сторінки. Без використання HTML-тегів браузер виведе невідформатований текст, без відступів, заголовків, абзаців тощо. Розглянемо структуру стандартної HTML-сторінки.

Сторінка завжди починається з відкриваючого тега <html> та закінчується закриваючим тегом </html> і складається з двох обов'язкових блоків — голови (head) і тіла (body), які записуються послідовно.

У блоці <head> </head> зберігається службова інформація, призначена допомогти браузеру в роботі з даними. Тут розташовані мета-теги, які використовуються для зберігання інформації, призначеної для пошукових систем, а саме: опис сайта, ключові слова тощо. Інформація є зазвичай невидимою для пересічного користувача, крім тега <title>, в якому відображається назва сторінки сайта.

Увесь контент, який відображається на сторінці, розміщується між відкриваючим і закриваючим тегами <body>.

Кожен абзац починається з тега <p> (від англ. *paragraph*).

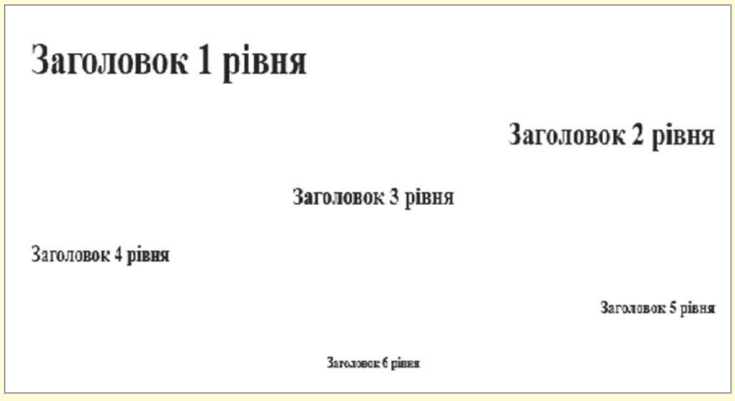
Зупинимось на тегах для роботи з текстом, який відображається у вигляді заголовків, абзаців і списків (рис. 11.35).

У HTML існують шість рівнів заголовків. Найвищим є заголовок першого рівня, найменшим — шостого. Позначаються вони відповідно <h1>… <h6>. Заголовки мають атрибут align, який визначає тип вирівнювання на сторінці та може набувати одне з чотирьох значень:

left	За лівим краєм (за замовчуванням)
right	За правим краєм
center	По центру
justify	За шириною (за лівим і правим краями)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
  <h1> Заголовок 1 рівня </h1>
   <h2 align='right'> Заголовок 2 рівня </h2>
   <h3 align='center'> Заголовок 3 рівня </h3>
   <h4 align='left'> Заголовок 4 рівня </h4>
   <h5 align='right'> Заголовок 5 рівня </h5>
   <h6 align='center'> Заголовок 6 рівня </h6>
</body>
</html>
```

Код HTML



Відображення браузером

Рис. 11. 35. Приклад запису заголовків та їх відображення браузером

Абзаци виділяються парним тегом   (від англ. *paragraph*). Як і теги заголовків, тег paragraph має атрибут align.

Списки можуть бути впорядковані, тобто нумеровані, та невпорядковані — марковані й багаторівневі.

Кожен елемент списку виокремлюється парним тегом <li>. Нумерований список позначається парним тегом <ol> (скор. ordered list — впорядкований список), маркований список — парним тегом <ul> (скор. *unordered list* — невпорядкований список) (рис. 11.36).

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
 <ol>
  <li>Елемент 1</li>
  <li>Елемент 2</li>
  <li>Елемент 3</li>
  <li>Елемент 4</li>
  <li>Елемент 5</li>
 </ol>
</body>
</html>
```

Код HTML

1. Елемент 1
2. Елемент 2
3. Елемент 3
4. Елемент 4
5. Елемент 5

Відображення браузером

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
 <ul>
  <li>Елемент списку 1</li>
  <li>Елемент списку 2</li>
  <li>Елемент списку 3</li>
  <li>Елемент списку 4</li>
  <li>Елемент списку 5</li>
 </ul>
</body>
</html>
```

Код HTML

- Елемент списку 1
- Елемент списку 2
- Елемент списку 3
- Елемент списку 4
- Елемент списку 5

Відображення браузером

Рис. 11.36. Приклад кодів нумерованого та маркованого списків та їх відображення браузером

У HTML існує спеціальний тег-контейнер для реалізації навігації по сайту — <nav>. Це парний тег, який містить навігаційні посилання (рис. 11.37).

Для посилання на іншу сторінку використовують парний тег <a> (скор. від англ. *Anchor* — якір). Тег має обов'язковий атрибут href, значенням якого є назва сторінки з розширенням .html.

Між відкриваючим і закриваючим тегами розміщується посилання на ресурс. Натиснувши на нього, користувач може переходити до потрібної сторінки.

З одного боку, HTML5 — це нова версія мови HTML, із новими елементами, атрибутами та новою поведінкою, з іншого — сукупність технологій, котра дозволяє створювати різноманітні сайти й веб-застосунки.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
  <nav>
    <a href="#">Головна</a>
    <a href="info.html">Інформація</a>
    <a href="gallery.html">Галерея</a>
    <a href="form.html">Зв'язатись з нами</a>
  </nav>
</body>
</html>
```

Код HTML

Головна Інформація Галерея Зв'язатись з нами

Відображення браузером

Рис. 11.37. Приклад запису навігаційного контейнера та його відображення браузером

Мова HTML постійно розвивається. Якщо ранні версії HTML поєднували функції розмітки та форматування контенту, тобто зміст (семантичний) та зовнішній вигляд цього змісту (презентаційний), то з появою багатосторінкових сайтів переважну частину функцій форматування було покладено на каскадні таблиці стилів (§ 11.8).

Наразі Консорціум просуває версію HTML5, це остання версія стандарту HTML. HTML5 остаточно відмовляється від використання презентаційних тегів, натомість з'являється низка нових елементів і атрибутів, які відображають типову архітектуру сучасних веб-сторінок. Приклад використання тегів HTML5 наведено на рис. 11.38.

У 1994 році засновано **Консорціум Всесвітньої павутини** — головну міжнародну організацію, що розробляє та впроваджує технологічні стандарти для Всесвітньої павутини.

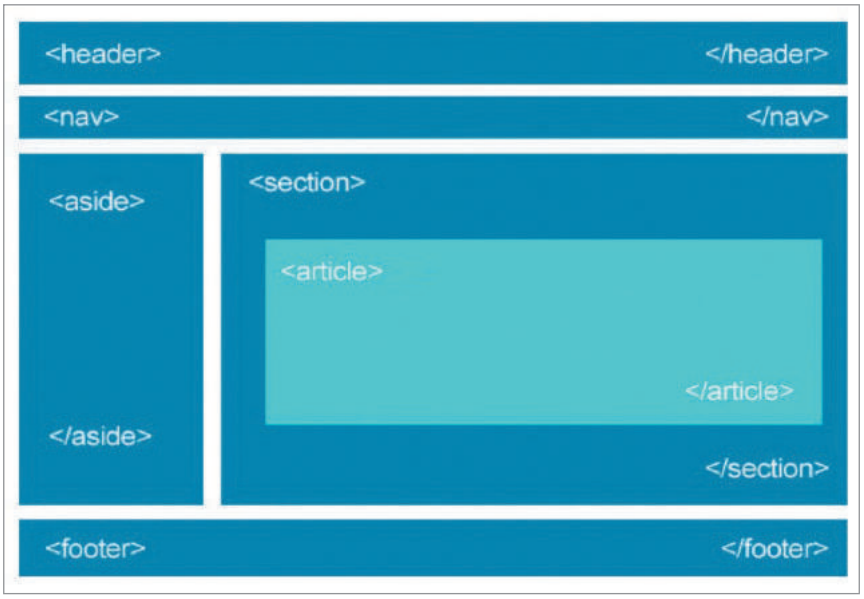


Рис. 11.38. Приклад розмітки сторінки з використанням тегів HTML5

Для більш глибокого вивчення технологій веб-розробки радимо скористатися курсом «Основи веб-розробки» освітньої платформи Прометеус. Викладач курсу Світлана Шабаранська, **Web-UI-девелопер** у компанії **N-iX**. Стане в пригоді веб-сайт для вивчення веб-технологій в Інтернеті (https://www.w3schools.com/), який містить навчальні посібники для вивчення **HTML**, **CSS**, **JavaScript** та ін.

Під час розроблення макета сайта завжди постає проблема заповнення макета текстом. Зазвичай для надання макету закінченого вигляду дизайнери застосовують так звану «рибу» — великий текстовий масив беззмістовних слів із потрібною кількістю символів, абзаців і параграфів, який так і називається — Lorem Ipsum.

Основна функція Lorem Ipsum полягає в зосередженні уваги на дизайні, а не на читанні вмісту. Ще одна причина, через яку використовують текст, — це заповнення сторінки для досягнення реального розподілу букв і пропусків у тексті, яке неможливе у випадку простого дублювання вислову «Тут написано ваш текст ... Тут написано ваш текст ...»

У більшість текстових і HTML-редакторів Lorem ipsum входить як текст за замовчуванням. Існують і сайти, на яких можна згенерувати потрібну кількість абзаців. Одним з найвідоміших і найпопулярніших сайтів-генераторів є Lipsum generator (https://uk.lipsum.com/) (рис. 11.39).

Рис.11.39. Головна сторінка генератора беззмістовного тексту латиною Lorem ipsum

Запитання для перевірки знань

1. Що таке контент; HTML; гіпертекст?
2. Як називають команди мови розмітки?
3. Створіть класифікацію тегів HTML.
4. Які невід'ємні компоненти структури сторінки?
5. Порівняйте HTML4 і HTML5. Зробіть презентацію за результатами порівняння.
6. Знайдіть в Інтернеті відомості про Тіма Бернерс-Лі, зробіть коротке повідомлення.

Завдання для самостійного виконання

1. Увійдіть у редактор коду, встановлений на комп'ютері. Створіть файл index.html.
2. Наберіть наступну послідовність команд (рис. 11.40).
3. Як текст використовуйте абзаци, згенеровані генератором Lorem ipsum.
4. Збережіть файл.
5. Перегляньте результат браузером.
6. Відкрийте файл у редакторі коду. Приберіть список, залишивши посилання з тегами <a>. Збережіть модифікований файл.
7. Перегляньте результат браузером.
8. Проаналізуйте, як відобразив браузер змінений файл.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Моя перша сторінка</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <header >
            <h1 align='center'>Головна сторінка</h1>
            <h2 align='right'>Слоган</h2>
        </header>
        <nav>
          <ul>
            <li><a href="#">Головна</a></li>
            <li><a href="#">Інформація</a></li>
            <li><a href="#">Контакти</a></li>
          </ul>
        </nav>
        <main>
            <p>! ДОДАТИ ТЕКСТ LOREM</p>
            <p align='right'>! ДОДАТИ ТЕКСТ</p>
            <p align='center'>! ДОДАТИ ТЕКСТ </p>
            <p align='left'>! ДОДАТИ ТЕКСТ </p>
            <p align='justify'>! ДОДАТИ ТЕКСТ </p>
        </main>
        <footer>
            <p>Copyright Example 2019</p>
        </footer>
    </body>
</html>
```

Рис. 11.40. До завдання 2

11.8. Каскадні таблиці стилів

Уявіть, що ви розробили сайт, що містить десять сторінок, у якому необхідно змінити колір тла і розмір шрифту та змінити вигляд меню. Ваші дії?

У 1994 році норвезький учений Хокон Віум Лі запропонував концепцію відділення дизайну сайта від його вмісту за допомогою каскадних таблиць стилів.

Стрімкий розвиток Всесвітньої павутини та поява величезної кількості сайтів у 1990-х роках спонукали до розробки нових тегів мови HTML, які відповідали за дизайн сайтів. Проте при зміні дизайну сайта в кожну його сторінку потрібно було вносити зміни, що значно ускладнювало роботу. У 1994 році було розроблено каскадні таблиці стилів.

Каскадні таблиці стилів (англ. *Cascading Style Sheets*, або скорочено CSS) — спеціальна мова, що використовується для опису зовнішнього вигляду сторінок, написаних мовою розмітки даних. Основна ідея CSS полягає в тому, щоб відокремити дизайн документа від його вмісту. CSS відповідає за оформлення і зовнішній вигляд HTML-коду, тоді як HTML — за зміст та логічну структуру документа.

Конструкція CSS, яка відповідає за зовнішній вигляд певного елемента HTML, називається CSS-правилом. Усі CSS-правила складаються із селектора та блоку оголошень.

Блок оголошень містить одне або кілька оголошень, розташованих у фігурних дужках. Усередині блоку оголошень знаходяться пари CSS-властивість — значення, розділені крапкою з комою (рис. 11.41).

Рис. 11.41. Правило оголошення CSS

З розвитком Всесвітньої павутини з'явилося безліч ідей оформлення веб-сторінки. Проте, на думку Бернерса-Лі, це не було надважливим завданням. Він розглядав веб насамперед як цифровий каталог наукових робіт (нагадаємо для чого створювалась мова HTML). Замість одного стандарту в той час існувало кілька конкуруючих мов для макетування веб-сторінок, створених різними розробниками, серед яких були Пей-Ян Вей, Андріссен і Хокон Віум Лі.

Кожне правило починається із селектора (покажчика), що вказує на ті HTML-елементи, до яких застосовується CSS-правило. Саме в блоці оголошень установлюються правила відображення вибраних нами елементів, визначаються їхні властивості — розмір, колір, відступи, поля, розташування на екрані (позиціонування) тощо. Селектор дозволяє звернутись до одного або кількох HTML-елементів.

Якщо необхідно визначити стиль таким чином, щоб один і той самий елемент у різних випадках відображався по-різному, то на допомогу приходять класи. Клас описується у вигляді .ім'я_класу {властивості}. Для присвоєння класу заданому тегу використовується властивість class = "ім'я_класу".

Ідентифікатори (селектори id) дуже подібні до класу, крім одного — ідентифікатор може мати одне-єдине унікальне ім'я в усьому документі. У файлі CSS ім'я вказується зі знаком решітки на початку, а до потрібного елемента додається атрибут id = " ".

Термін «*cascading*» (каскадні) у назві CSS вказує на можливість злиття різних таблиць стилів для створення єдиного визначення стилю окремого елемента (тегу) чи всього документа.

Каскадність CSS — це механізм, завдяки якому до елемента HTML-документа може застосовуватися більш ніж одне правило CSS.

Правила можуть виходити з різних джерел: із зовнішньої та внутрішньої таблиці стилів, від механізму наслідування, від батьківських елементів, від класів і id, від селектора тегу, від атрибута style тощо. Оскільки в цих випадках часто відбувається конфлікт стилів, була створена система пріоритетів: у кінцевому підсумку застосовується той стиль, який виходить від джерела з більш високим пріоритетом.

Таким чином, каскадність у CSS — це здатність стильових правил накладатися один на одного, перезаписувати і змішуватися. Підсумковий стиль елемента, який видно в браузері, — це комбінація кількох послідовно застосованих стилів.

Існує кілька способів підключення CSS-коду до HTML-документа. Розглянемо деякі з них.

- Застосування inline-стилів (стилі, які підставляються безпосередньо в рядок).

Додавання CSS-правила в HTML-тег за допомогою атрибута style. Усередині атрибута style можна написати кілька CSS-оголошень, розділених крапкою з комою, фігурні дужки не використовуються. Inline-стилі змішують уміст документа і його дизайн, тому його краще використовувати як виключення, у випадку, коли елемент зустрічається лише один раз у документі або на сайті, але вимагає особливого оформлення (приклад 1).

- Застосування таблиць стилів документа (*document style sheets*).

Називаються так тому, що розташовуються безпосередньо в HTML-документі й застосовуються лише до нього. Іноді їх називають embedded style sheet (убудований стиль). Збереження CSS-коду в HTML-документі у тезі <style>, що теж розміщується в <head>. Зазвичай цей варіант використовується, коли існує лише одна проста HTML-сторінка й немає сенсу створювати додатковий файл (приклад 2).

- Застосування зовнішніх, або зв'язаних, стилів (*external style sheets*).

Це найбільш поширений варіант. Він полягає у винесенні CSS-коду в окремий файл із розширенням .css та підключенням за допомогою тегу <link>, який знаходиться виключно всередині елемента <head>. Зустрівши в HTML-документі цей тег, браузер завантажить із сайта CSS-файл і застосує до документа стилі, що містяться в ньому (приклад 3).

Існує ще один спосіб підключення таблиць — за допомогою директиви @import. Цей спосіб дозволяє об'єднувати кілька таблиць стилів в одну (приклад 4).

Приклад 1.

```
<body>
<header style="background-color:grey;
            width: 900px;
            height: 100px;
            text-align:center;
            padding-top: 30px ;
            margin: 10px auto;">
Заголовок сайта
        </header>
</body>
```

Приклад 2.

```
<head>
<style type="text/css" >
header {
            background-color: grey;
             width: 900px;
             height: 100px;
             text-align: center;
             padding-top: 30px ;
             margin: 10px auto;
            }
</style>
<body>
<header>  Заголовок сайта </header>
</body>
```

Приклад 3.

```
<head>
<link rel="stylesheet" type="text/css" href="css/style.css">
</head>
<body>
<header >
      Заголовок сайта
</header>
</body>
```

HTML код

```
header {
        background-color: grey;
        width: 900px;
        height: 100px;
        text-align: center;
        padding-top: 30px ;
        margin: 10px auto;
      }
```

Файл style.css

Приклад 4.

```
<head>
<style type="text/css">
@import url("css/style.css")
</style>
<body>
<header>  Заголовок сайта
</header>
</body>
```

Правила форматування CSS

- Кожна властивість — в окремому рядку.
- Кожен селектор — в окремому рядку.

Порядок опису стилів

- Позиціонування (position, left/right/top/bottom, float, clear, z-index.)
- Дісплей і блочне моделювання (display, width, height, margin, padding…
- Колір (background)
- Текст (list-style-type, overflow, color font…)
- Інші

Свого часу цей спосіб був досить популярний, проте наразі він втрачає свої позиції через те, що сторінка не завантажиться, доки браузер не завантажить файл CSS повністю. Це негативно позначається на швидкості завантаження сайта й відповідно гальмує роботу.

Розглянемо пріоритетність стилів.

1) Якщо в таблиці є однакові селектори, то виконуватиметься той, який записаний останнім.

Наприклад p{color: grey;} p{color: pink;} — колір абзацу буде рожевим.

2) Якщо для одного елемента задано стиль і в зовнішній, і у внутрішній таблицях, то пріоритет віддається стилю в тій таблиці, яка знаходиться нижче в коді. Наприклад, спочатку в <head> підключили зовнішню таблицю, а потім за допомогою тегу <style> додали внутрішню таблицю. Браузер відобразить стилі внутрішньої таблиці.

3) Пріоритетність за спаданням:
   - inline-стилів,
   - ідентифікаторів,
   - класів,
   - тегів.

Ознайомимося зі стислим описом модулів CSS.

№ з/п	Назва модуля	Опис
1	CSS-позиціонування	CSS-позиціонування (positioning) дозволяє вказати, де з'явиться блок елемента, а вільне переміщення (floating) переміщує елементи до лівого або правого краю блока-контейнера, дозволяючи решті вмісту «обтікати» його
2	CSS-текст	Керують перетворенням вихідного тексту у форматований і перенесенням рядків
3	CSS-шрифт	Вибір шрифту (font-family), його насиченість (font-weight), ширина (font-stretch), накреслення (font-style), розмір (font-size)

Закінчення таблиці

4	CSS-посилання	Містять властивості, які відповідають за зовнішній вигляд гіпертекстових посилань HTML-документа. Оскільки посилання є основним способом навігації сайтом, то застосування CSS-стилів для оформлення покращить їх візуальне сприйняття
5	CSS-списки	CSS-списки — набір властивостей, які відповідають за оформлення списків. Зазвичай списки використовуються під час створення навігаційних панелей сайта. За допомогою стандартних CSS-властивостей можна змінити зовнішній вигляд маркера списку, додати зображення для маркера, а також змінити місце розташування маркера
6	CSS-тло	CSS-тло — властивості, що додають тло для будь-якого HTML-елемента. Кожен елемент має шар тла, який може бути прозорим (за замовчуванням), мати кольорове заповнення та зображення
7	CSS-color	Описує значення, які дозволяють визначати колір та непрозорість HTML-елементів, а також значення властивості color

Запитання для перевірки знань

1. Що таке каскадні таблиці стилів? Назвіть причини їхньої появи.
2. Опишіть синтаксис CSS-правила.
3. Що таке селектор? Які бувають селектори?
4. Наведіть пріоритети виконання таблиць стилів.
5. Опишіть способи підключення стилів. Проаналізуйте, коли і який спосіб краще використовувати.
6. Виберіть із наведеної раніше таблиці CSS-модуль. Розробіть презентацію з докладним розбором та прикладами використання правил CSS.

Завдання для самостійного виконання

1. Увійдіть у редактор коду, встановлений на вашому комп'ютері, і створіть файл style.css.
2. Наберіть наведену послідовність команд (рис. 11.42).
3. Збережіть файл у тій самій теці, де ви зберегли файл index.html, створений на попередньому уроці.
4. Відкрийте файл index.html у редакторі коду. Додайте у блок <head> таку команду.
5. <link rel="stylesheet" type="text/css" href="style.css">
6. Збережіть модифікований файл.
7. Перегляньте результат браузером.
8. Проаналізуйте, як відобразив браузер змінений файл.

```
body{
        font-family: "Segoe UI";
        background-color: #153e5c;
}
h1,h2{
        text-align: center;
        color: #cdd4ca;
}
a{
        color: #cdd4ca;
        text-decoration: none;
}
a:hover{
        color: #f2be54;
}
li{
        list-style: none;
        display: inline-block;
}
p{
        font-size: 150%;
        color: white;
}
```

Рис. 11.42. До завдання 2

11.9. Проектування та верстка веб-сторінок

Поміркуйте, чим відрізняються поняття сайта й веб-сторінки.

Рис. 11.43. Порядок дій створення сайта

Процес створення сайта (веб-проекту) умовно можна розподілити на кілька етапів (рис. 11.43). Розглянемо такі етапи, як загальне планування; розробка дизайну; планування макета веб-сторінок.

Загальне планування складається зі створення ідеї, розробки структури проекту, опрацювання макета проекту.

Створення ідеї: необхідно вибрати тему проекту (сайта сервісу) й відповідно до неї дібрати текстові та графічні матеріали.

Розробка структури проекту: потрібно визначити кількість розділів сайта, класифікувати матеріал за розділами, приступити до формування навігаційного меню.

Опрацювання макета проекту: потрібно скласти макет проекту, використовуючи графічний редактор. У 10 класі в розділі растрової графіки ви вже вивчали графічний редактор Gimp, який має потужний інструментарій для розробки макета сайта, створення банерів, кнопок і логотипів, опрацювання фотографій, створення gif-анімації. Можна схематично скласти макет проекту, використовуючи також папір і ручку.

Схематичний начерк дає розуміння того, як на сайті розташовуватимуться основні інформаційні блоки, графічні зображення тощо (рис. 11.44).

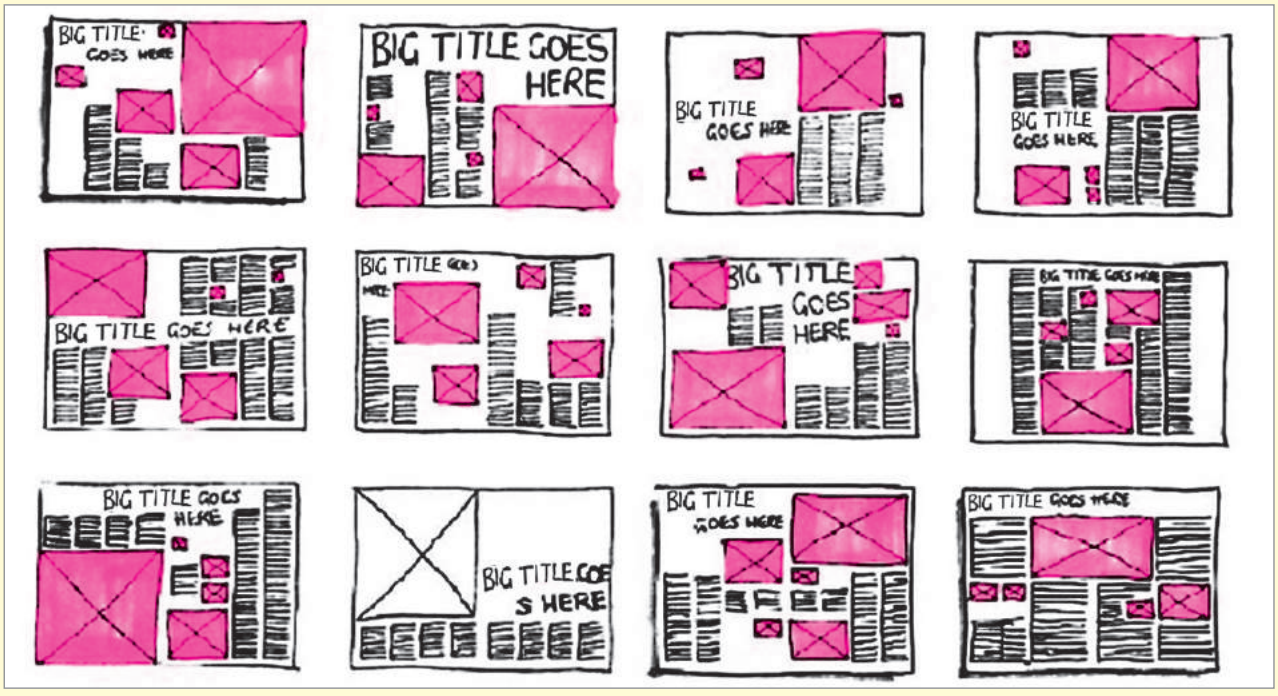


Рис. 11.44. Вибір макета проекту

Переважна більшість сучасних сайтів має блочну верстку. Розглянемо основні елементи веб-сторінки сайта: блок (або контейнер — div), заголовок (хедер), у який входять назва сайта та логотип, навігація, контент, нижній колонтитул (футер, підвал). Звернемося до історії розвитку верстки.

Веб-сторінки спочатку були просто набором тегів. Елементи відображалися в тому порядку, в якому записувалися в HTML-коді. Щоб візуально розділити інформацію, використовувалися горизонтальні лінії або відступи.

Кроком вперед була так звана *таблична* верстка веб-сторінки: елементи структурувалися за допомогою таблиці, інформація вставлялась у клітинку. Створити сайт за допомогою таблиці було досить просто — достатньо мати початкові знання HTML і CSS і використовувати мінімум правил CSS.

Таблична верстка була дуже популярною на початку 2000-х років. До сьогодні існує досить багато сайтів, які зверстано у вигляді таблиць.

Таблична верстка має низку недоліків: при складній структурі важко розібратися в коді, браузери відображають таблицю на екрані лише після повного завантаження, а складний дизайн з перекриттям елементів узагалі неможливо реалізувати. Таблична верстка неприйнятна для побудови адаптивних сайтів.

Наразі найпопулярнішим способом верстки є так званий *блочний*. При цьому використовуються блоки, які або розташовують один під одним, або керують порядком їх відображення за допомогою позиціонування в CSS.

Розглянемо основні елементи веб-сторінки сайта.

Елемент div() (від англ. *division* — розділ, блок) зазвичай виконує на веб-сторінці роль контейнера (рис. 11.45).

Теги div мають такі характеристики:

- div — блочний елемент; якщо ширину не задано, блок займає всю ширину браузера;
- div — висота блоку; якщо висоту не задано, то блок дорівнює вмісту; порожній блок div має висоту 0 px, тому не відображається на сторінці;
- div не має оформлення; для того щоб його побачити, потрібно задати стилі в CSS;
- div — може містити будь-яке число вкладених елементів; можна вкласти інші блоки div, заголовки, таблиці, зображення та ін.

Елемент <header> (заголовок) є контейнером, у якому містяться назва сайта, логотип і навігаційна панель.

Логотип найчастіше розташовується у верхньому лівому кутку веб-сторінки або посередині, залежно від ідеї, макета (рис. 11.46).

Навігаційна панель часто розташовується у верхній частині веб-сторінки незалежно від того, вертикально або горизонтально розташовуються елементи навігації сайта (рис. 11.47), і містить посилання на його основні розділи.

Нижній колонтитул (*footer*, підвал) розташовується внизу веб-сторінки і зазвичай містить інформацію про правовласників, контактні та юридичні дані, посилання на основні розділи сайта (найчастіше дублює основну навігацію), посилання на соціальні мережі, форму зворотного зв'язку та ін. (рис. 11.48).

Одиниці вимірювання в CSS:

- піксель (px) — абсолютна одиниця вимірювання, оскільки не змінюється залежно від інших налаштувань;
- em — масштабована одиниця, яка вказує відношення до розміру шрифту (font-size) поточного елемента (якщо font-size у документі 12pt, тоді 1em = 12pt, 2em = 24pt і т. д.);
- vw і vh — дозволяють вказувати розміри відносно вікна користувача (1vw — це 0,01 ширини вікна, а 1vh — 0,01 висоти).

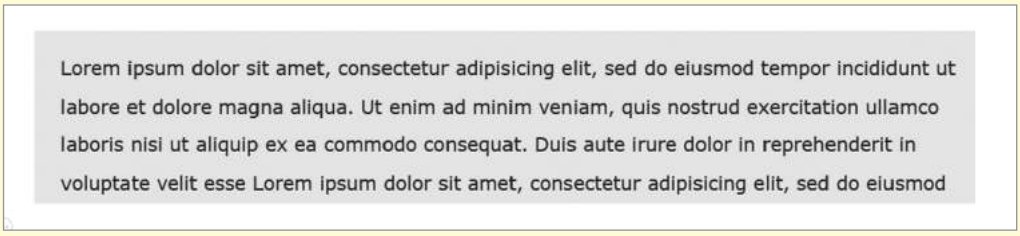
Контент є основною складовою веб-сторінки. Він відіграє провідну роль у дизайні сайта, тому займає більший простір, крім тексту, підкріплений графікою.

```
<div class="text">
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
</div>
```

а

```
.text {
        margin-top: 3vh;
         width: 70%;
         margin-left: auto;
         margin-right: auto;
         height: 16vh;
         overflow: hidden;
         line-height: 4vh;
         color: #303030;
         font-family: Verdana;
         font-size: 2vh;
         transition: 0.4s height;
}
```

б



в

Рис. 11.45. Приклад використання контейнера на веб-сторінці: код HTML (*а*); правила CSS (*б*); відображення в браузері (*в*)

```
<a class='w3schools-logo notranslate'
href='//www.w3schools.com'>w3schools<span
class='dotcom'>.com</span></a>

.w3schools-logo {
font-family:fontawesome;
text-decoration:none;
line-height:1;
font-size:37px;
letter-spacing:3px;
color:#555555;
display:block;
position:absolute;
top:17px;
}
w3schools-logo .dotcom
{color:#4CAF50}
```

w3schools.com

Рис. 11.46. Приклад запису та відображення у браузері логотипа сайта

```
<nav class="menu">
        <li><a href="#" >Головна</a></li>
        <li><a href="#" >Галерея</a></li>
        <li><a href="#">Контакти</a></li>
</nav>
```

```
menu {
        display: inline-block;
        width: 100vw;
        height: 10vh;
        background-color: lightgreen;
        align-items: center;
        justify-content: center;
        margin-top: 35vh;
        padding-left: 18vw;
        padding-right: 18vw;
}
.menu li { list-style: none;}
```

Рис. 11.47. Навігаційна панель сайта

```
<footer>
    ©All Rights Reserved
 <br>
   Знайдіть нас у мережах:
   <a href="google.com">Google</a>
   <a href="youtube.com">YouTube</a>
</footer>
```

КОД HTML

```
footer{
        background:
#b6e283;
        width: 100%;
        padding: 1%;
}
```

ПРАВИЛА CSS

Рис. 11.48. Нижній колонтитул веб-сторінки

Модульна сітка передбачає поділ веб-сторінки на окремі колонки по вертикалі та вибудовування контенту. Дизайн макета зазвичай розробляється саме за цією сіткою (рис. 11.49).

Найбільш популярною є модульна сітка 960 Grid System (http://960.gs), яка максимально ділить сторінку на 12, 16 і 24 колонки. За шириною сітка становить максимум 960 пікселів. Такий вибір базується на тому, що на момент створення сітки більшість сучасних моніторів мали роздільність 1024×768 пікселів.

Створення макета на основі модульної сітки допоможе в подальшому прискорити процес верстання. Завдяки їй блоки контенту й елементи розташовуватимуться на певній відстані один від одного.

Можна сказати, що модульна сітка — це певна візуальна абстракція, візуальний розподіл сторінки на однакові за шириною стовпці з однаковими відступами між ними. Візуалізувати модель можна за допомогою напрямних або окремого шару, на якому будуть зображені ці стовпці (знову пригадаємо графічний редактор Gimp).

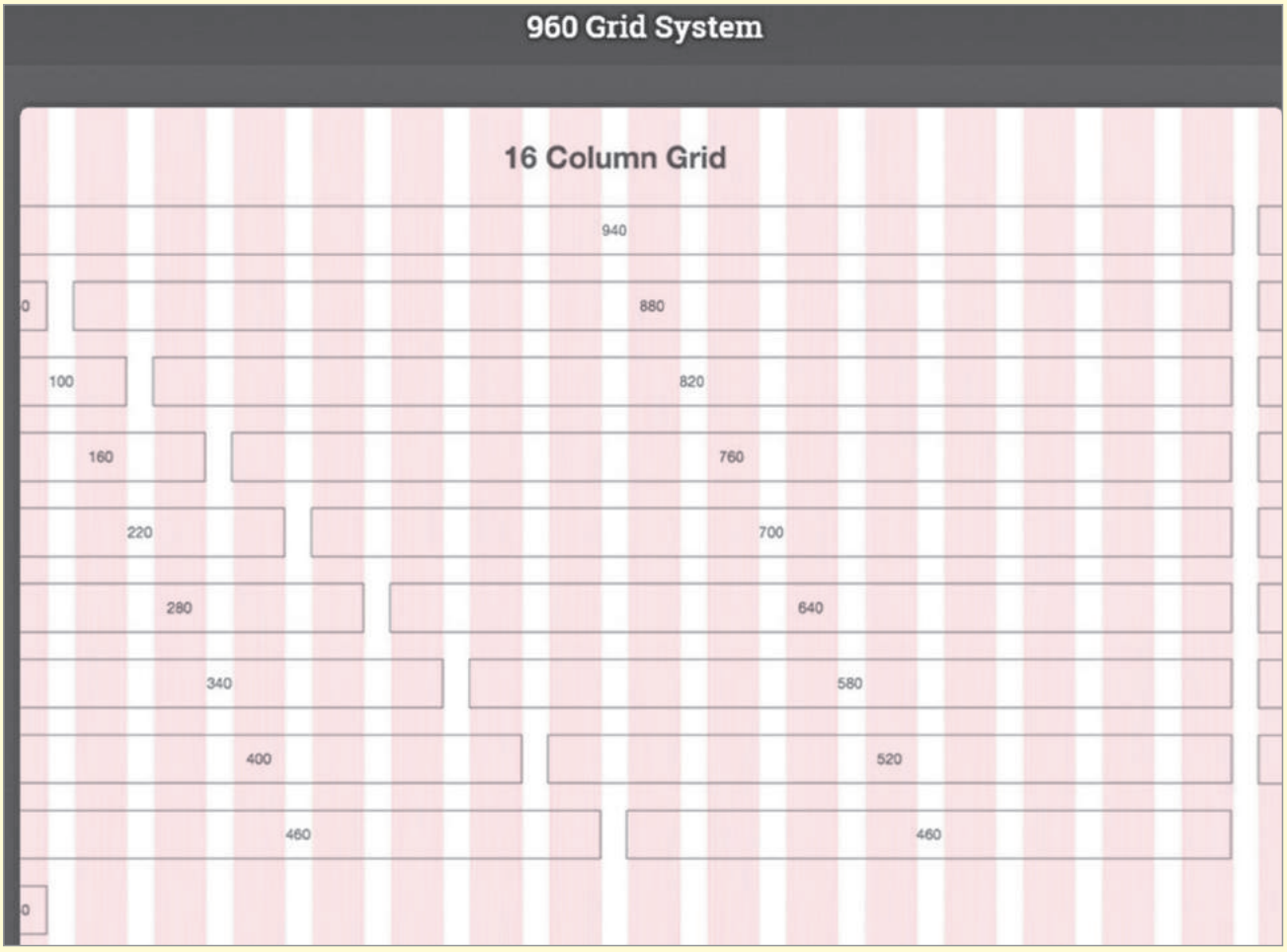
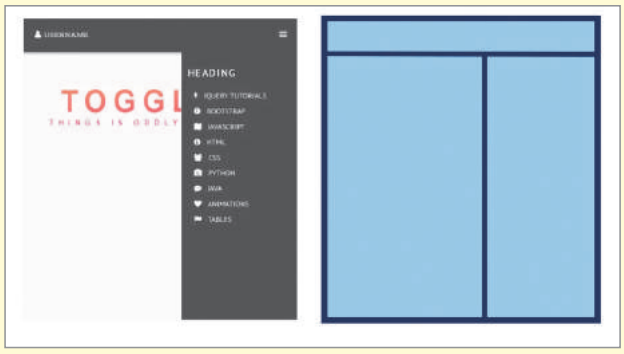


Рис. 11.49. Модульна сітка

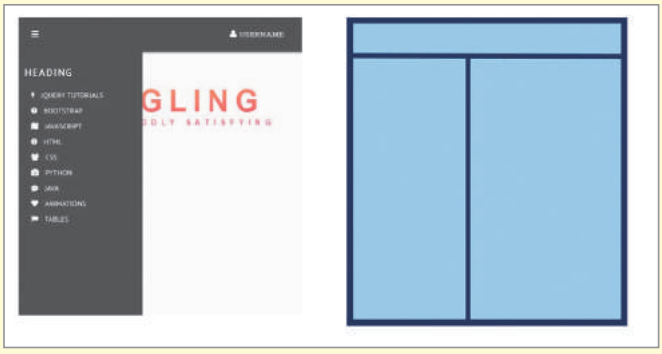
Серед усього різноманіття складання макета веб-сторінки можна виділити різні типи навігації.

Розглянемо чотири найбільш поширені типи навігації вебсторінкою сайта (рис. 11.50):

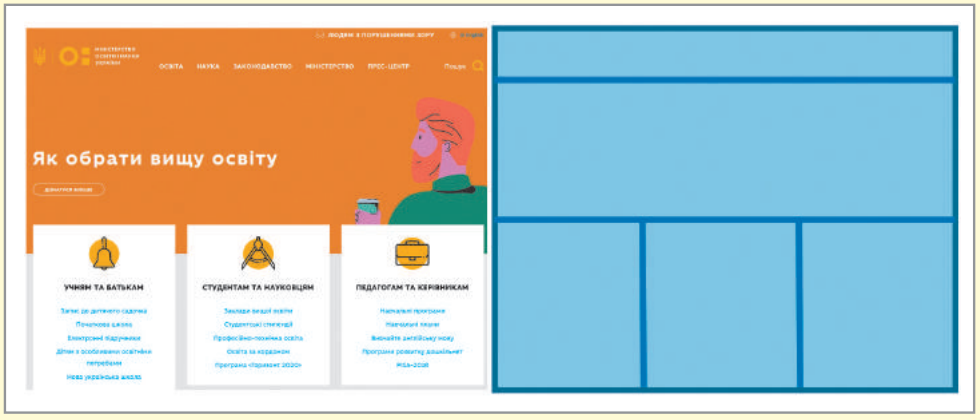
- навігація в лівому стовпці (випадок *а*);
- навігація в правому стовпці (випадок *б*);
- горизонтальна навігація (випадок *в*);
- навігація Mobile First (випадок *г*).



а



б



в

г

Рис. 11.50. Приклади сайта: з навігацією в лівому стовпці (*а*); навігацією в правому стовпці (*б*); горизонтальною навігацією (*в*); технологією **Mobile First** (*г*)

Сайти з горизонтальною навігацією становлять більшість. Такий вибір пояснюється зручністю, адже залишається більше простору для контенту, яким наповнено сайт.

З урахуванням тенденцій останніх років тип навігації Mobile First впевнено займає свою нішу. У разі його використання розробка макета сайта, дизайну й верстки починається з мобільної версії, а вже потім опрацьовуються макети для інших роздільностей: додаються блоки, банери, додаткові елементи дизайну тощо (більш детально цей підхід описано в § 11.10).

Одним зі способів визначення основного кольору в проекті є складання mood board (у перекладі з англійської мови — дошка настрою). Під час створення дизайну макета варто розпочати роботу з визначення колірної гами проекту.

Порядок складання mood board такий: потрібно набрати кожен синонім у рядку пошуку по картинках Google та виписати найчастіше повторювані в знайдених зображеннях кольори, — від них залежить візуальне сприйняття проекту користувачем і виклик відповідних почуттів.

Понад 80 % користувачів Інтернету використовують для доступу в мережу мобільні пристрої, тож правилом хорошого тону стає розробка не тільки десктопної, а й мобільної версії сайта.

Для роботи з вибраним кольором і складанням палітри кольорів сайта можна використовувати сервіси Color Scheme Designer 3 і Adobe Color CC.

Color Scheme Designer 3 (http://colorschemedesigner.com/csd-3.5/) надає можливість пперегляну ти, який вигляд матиме сайт у вибраних кольорах (рис. 11.51).

Рис. 11.51. Палітра кольорів **Color Scheme Designer**

Adobe Color CC (https://color.adobe.com) дозволяє створювати палітри ще й на основі завантажених зображень (які, наприклад, могли з'явитися в нас під час складання mood board). Сервіс містить великий архів палітр інших користувачів (рис. 11.52).

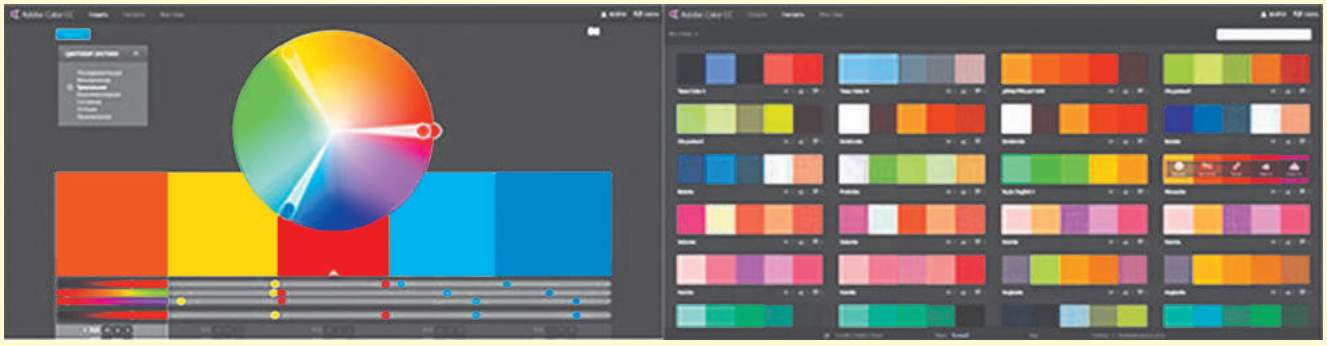


Рис. 11.52. Палітра кольорів **Adobe Color CC**

Під час розробки дизайну веб-сторінки використовують фреймворк — програмну оболонку, що дозволяє спростити і прискорити розв'язування типових завдань.

Досить часто використовують фреймворки Bootstrap, Foundation, Material Design Lite. Крім готових елементів дизайну (кнопки, форми введення тощо) вони пропонують свою модульну сітку, CSS-сніппети (частина коду, розмітки, яка

може неодноразово використовуватися) для вставки елементів у веб-сторінку (кнопок, елементів форм та ін.) і класи розмітки, а так само JS-скрипти для відповідних інтерактивних елементів.

На основі певного фреймворка можна знайти величезну кількість платних і безкоштовних тем і сторінок, а також розробити власні.

Під структурою проекту розуміють зберігання файлів проекту в його директорії. Окремі категорії файлів необхідно поміщати у свої папки: картинки в папку images або img, css — у папку css, javascript — у папку js. У корені лежатимуть лише index.html і веб-сторінки сайта, або тільки index.html, а веб-сторінки — в окремій папці pages.

Важливо дотримуватись подібних правил і під час іменування файлів проекту.

Найчастіше застосовуються такі імена, як головна сторінка — index.html, стилі проекту — styles.css, скрипти — scripts.js або app.js. Мінімізовані версії файлів мають префікс .min, назви картинок відображають те, що на них зображено.

Верстання веб-сторінки здійснюється поетапно. Порядок дій під час верстання веб-сторінок наведено на рис. 11.54.

Як бачимо, за допомогою тегів HTML спочатку створюють скелет сторінки, орієнтуючись на структуру, складену ще на першому етапі. Далі визначають необхідні класи та переходять до написання CSS-стилів.

Заключним етапом є написання JS-скриптів (розглядатиметься в § 11.15–11.16).

Після написання HTML, CSS і JS для сторінки необхідно перевірити, чи все зроблено правильно (розглядатиметься в § 11.20)

Рис. 11.53. Структура теки **Site**

Рис. 11.54. Етапи верстання веб-сторінки

Запитання для перевірки знань

1. Сформулюйте правила структурування й іменування файлів.
2. Які типи макетів веб-сторінок ви знаєте? Які переваги та недоліки вони мають?
3. Опишіть етапи створення сайта.
4. Яке місце в дизайні сайта, на вашу думку, посідає колірна гама?
5. Чим корисні фрейворки?
6. Опишіть етапи верстки веб-сторінки.

Завдання для самостійного виконання

1. Створіть теку для вашого майбутнього сайта з назвою Site.
2. У створеній теці задайте ще відповідно теки images, css, js.
3. Перенесіть у теку Site файл index.html, модифікований на минулому уроці.
4. У теку css перенесіть файл style.css.
5. У файлі index.html виправте посилання на файл style.css так, щоб ви могли знайти його у новостворені теці. Для цього у запису <link rel = "stylesheet" type = "text/css" href = "style.css"> пропишіть шлях до файла наступним чином href = "css/style.css".
6. Перегляньте отриманий результат у браузері.

11.10. Адаптивна верстка

Чи користуєтеся ви мобільним Інтернетом? Як ви гадаєте, чи однаково виглядає код сайта для перегляду на стаціонарному комп'ютері та смартфоні?

Джерела інформації повсякчас урізноманітнюються й удосконалюються. Ознайомимося з ними докладніше.

Із появою мобільного Інтернету до класичного веб-сайта додався так званий вап-сайт. Перший, як відомо, призначений для перегляду на комп'ютері, другий — для перегляду з мобільного телефону. Погодьтеся, це додало зручності. Адже веб-сайти, маючи широкий функціонал і яскраве оформлення, дуже довго завантажувалися й забирали занадто багато трафіку. Натомість вап-сайти містили в основному текст і поодинокі зображення. Додамо, що на той час мобільні телефони були слабкі, а мобільний Інтернет повільний і дуже дорогий.

Згодом широким попитом стали користуватися планшети. Це створило нові виклики фахівцям у галузі веб-дизайну, адже фіксована верстка не дозволяла змінювати розмір роздільності сайта. Великі сайти переглядати з планшетів було незручно, а мобільні сайти не були оптимізовані під екран великих розмірів. Сьогодні використання смартфонів, планшетів і Smart-TV є невід'ємною частиною сучасного життя, а кількість і різноманітність мобільних пристроїв постійно збільшується (рис. 11.55).

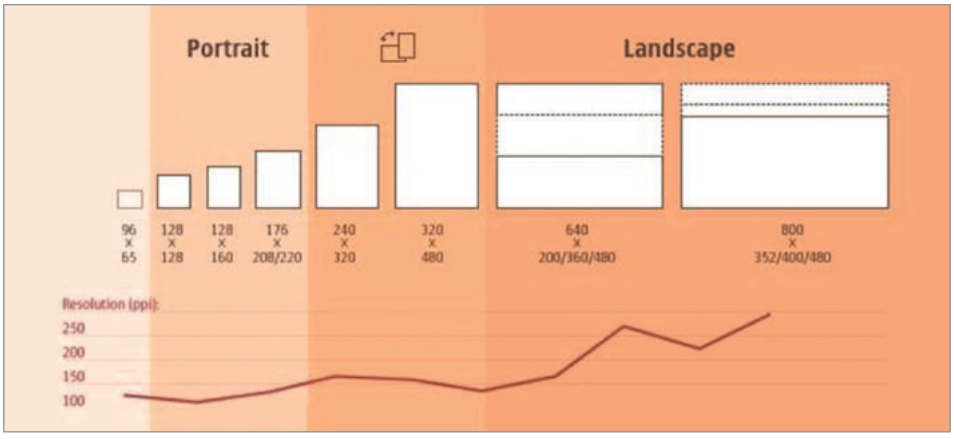
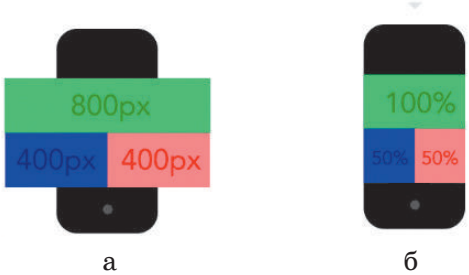


Рис. 11.55. Статистика різноманітних девайсів та їх роздільність

Першою спробою підлаштування сайта під розміри екранів стала так звана *гумова верстка*. У разі її застосування й макету, й окремим його складовим надаються розміри не фіксовані (рис. 11.56, *а*), а подані у відсотках (рис. 11.56, *б*), тобто вміст сторінки розтягується. Проте коли, наприклад, той самий контент розтягується на 6-дюймовий екран смартфона та 42-дюймовий екран телевізора, це вже заважає. Тож необхідно встановити максимальний і мінімальний розмір (властивість max-width).



а б

Рис. 11.56. Приклад відображення на смартфоні макета веб-сторінки: *а* — фіксована верстка; *б* — гумова верстка

Наступним кроком стала поява медіа-запитів. Цьому сприяла пропозиція Марка ван ден Доббельстіна щодо додавання класів під час завантаження та оголошення кожному діапазону спеціальних стилів.

Медіа-запити (*media queries*) — це правила CSS, які дозволяють керувати стилями елементів залежно від значень технічних параметрів пристроїв.

У 2001 році в HTML4 і CSS2 була введена підтримка апаратнозалежних таблиць стилів, що дозволило створювати стилі й таблиці стилів для певних типів пристроїв.

Таким чином, браузер застосовував таблицю стилів тільки в разі, якщо активувався саме цей тип пристрою:

У 2006 році в інтернет-журналі **A List Apart** (https://alistapart.com/) вийшла стаття Доббельстіна, а через 2 роки з'явилися медіа-запити.

Тип носія	Опис медіа-запитів
all	Всі типи. Це значення використовується за замовчуванням
braille	Пристрої, засновані на системі Брайля й призначені для читання людьми з вадами зору
embossed	Принтери, що використовують для системи друку шрифтом Брайля
Handheld	Смартфони та аналогічні пристрої
print	Принтери та інші друкарські пристрої
projection	Проектор
screen	Екран монітора
speech	Мовні синтезатори, а також програми для відтворення тексту вголос (сюди, наприклад, можна віднести мовні браузери)
tty	Пристрої з фіксованим розміром символів (телетайпи, термінали, пристрої з обмеженнями дисплея)
tv	Телевізори

Як же використовують медіа-запит? Загалом медіа-запит складається з ключового слова, що описує тип пристрою (необов'язковий параметр) і вираз, котрий перевіряє характеристики цього пристрою. З усіх характеристик найчастіше перевіряється ширина пристрою width.

Медіа-запит є логічним виразом, який повертає істину або хибність.

Запит записують у кінці таблиці стилів (пригадаємо правила запису стилів: у стилів, які записані пізніше, вища пріоритетність). Наприклад, наведений далі запис означає, що стилі будуть використовуватися лише у випадку, коли ширина екрану буде не більшою за 600 пікселів:

```
@media screen and (max-width: 600px) {
}
```

Розглянемо на прикладі використання медіа-запитів.

Приклад.

```
<style>
.block_left { width: 430px; }
@media (max-width: 1220px) {
 .block_right { width: 380px; }
}
@media (max-width: 1120px) {
 #block { width: 325px; }
}
@media (max-width: 680px) {
 #block { width: 200px; }
}
</style>
```

Ідея розробки адаптивної верстки належить Аарону Густафсону. У своїй книзі «Адаптивний веб-дизайн» він запропонував гнучко адаптувати сайти до можливостей пристроїв і браузерів.

Наведемо список популярних медіа-запитів для стандартного набору розширення екранів.

```
@media only screen and (max-width : 1920px) {/* CSS правила */ }
@media only screen and (max-width : 1680px) {}
@media only screen and (max-width : 1366px) {}
@media only screen and (max-width : 1280px) {}
@media only screen and (max-width : 1024px) {}
@media only screen and (max-width : 800px) {}
@media only screen and (max-width : 768px) {}
@media only screen and (max-width : 600px) {}
@media only screen and (max-width : 533px) {}
@media only screen and (max-width : 360px) {}
@media only screen and (max-width : 320px) {}
@media only screen and (max-width : 240px) {}
@media only screen and (max-width : 176px) {}
```

Адаптивні сайти з'явилися завдяки зростанню попиту тих користувачів, які хочуть завантажувати веб-сторінки з ідеальною структурою на всіх своїх пристроях. Саме зараз адаптивний дизайн стає обов'язковим. Він значно спрощує регулярний серфінг в Інтернеті незалежно від того, який пристрій вибрав користувач для доступу до сайтів, які його цікавлять.

Сьогодні прийнято вважати, що головне в адаптивній верстці — це прив'язка до конкретних пристроїв з певною роздільністю екрану. Стилі перемикаються від одного брейкпоінта (контрольні точки, сталий термін у програмуванні, у випадку верстки — перемикання з одних умов на інші) до іншого, тобто насправді є фіксовані макети для кожного девайсу.

Традиційно макет сайта спочатку розробляється для десктопної версії, а вже потім адаптується до мобільної. Люк Вроблевскі у своїй книзі «Спочатку мобільні» запропонував піти від протилежного: розробити макет сайта для мобільної версії, а вже потім покращувати до десктопної. Вроблевскі керувався тим, що ускладнювати просте легше, ніж спрощувати складне.

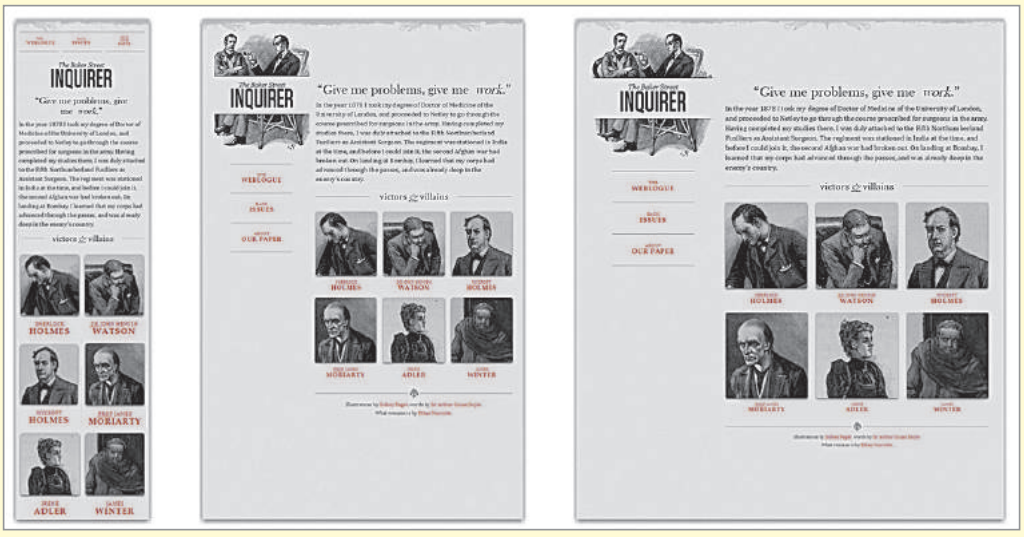


Рис. 11.57. Сайт Ітана Маркотта

За Аароном Густафсоном, адаптивність — це особливий підхід до розробки сайта, який дозволяє вже наявним веб-ресурсам підлаштовуватися під розміри екранів різних пристроїв. Інакше говорячи, сторінка повинна автоматично підлаштовуватися під екран, змінювати розмір картинок тощо. Це дозволило усунути потребу в розробці дизайну для кожного нового типу пристрою.

Основні особливості адаптивного дизайну:

- застосування гнучкого макета на основі сітки (англ. *flexible, grid-based layout*);
- використання гнучких зображень (англ. *flexible images*);
- робота з медіа-запитами (англ. *media queries*);
- плавна перебудова блоків у разі зміни розміру екрана (наприклад, під час повертання планшета)

Якщо зображення гнучке, це означає, що його розмір змінюється залежно від розміру контейнера, в якому воно міститься (§ 11.12).

Flexbox, CSS Grid і багатоколонкова верстка (*Multi-column layout*) є адаптивними за замовчуванням (їх специфікації були написані у світі, де адаптивний дизайн і крос-девайсність вже стали реальністю). Це означає, що ім притаманні безліч функцій, які дозволяють легко створювати адаптивні сітки.

CSS Flexbox (*Flexible Box Layout Module*) — модуль макета гнучкого контейнера, що являє собою спосіб компонування елементів. Технологія flexbox ставить на меті зробити шари гнучкими, а роботу з ними інтуїтивно зрозумілою.

Flexbox складається з гнучкого контейнера (*flex container*) і гнучких елементів (*flex items*). Останні можуть вибудовуватися в рядок або стовпець, а вільний простір розподіляється між ними різними способами.

Модуль flexbox дозволяє:

- розташовувати елементи в одному з чотирьох напрямків: зліва направо, справа наліво, зверху вниз або знизу вгору;
- перевизначати порядок відображення елементів;
- автоматично визначати розміри елементів так, щоб вони вписувалися в доступний простір;
- розв'язувати проблему з горизонтальним і вертикальним центруванням;
- переносити елементи всередині контейнера, не допускаючи його переповнення;
- створювати колонки однакової висоти.

Flex (від англ. *Flex* — розтягувати) і flex-inline — це параметри для властивості елемента display батьківського HTML-елемента, що містить дочірні блоки. На відміну від інших параметрів цього елемента (block || inline-block || table) вони не задають новий тип відображення елемента.

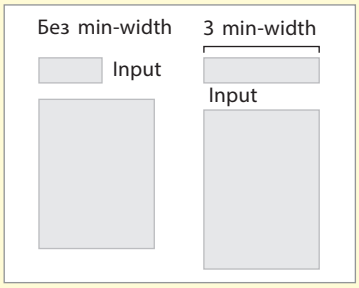
Вони надають можливість керувати поведінкою дочірніх елементів, вкладених у контейнер-обгортку з цією властивістю, наприклад, змінювати їх розмір і відстань між ними (рис. 11.59).

Книга Аарона Густафсона

Концепцію чуйного дизайну запропонував Ітан Маркотт у 2010 році та описав у своїй книзі «Responsive Web Design». Особливістю такого підходу стала плавна зміна сайта, зорієнтована на вміст, а не на конкретні пристрої.

Книги Ітана Маркотта та Люка Вроблевські

Найпростішим способом реалізувати змасшабовану сторінку є використання максимальної (max-width) й мінімальної (min-width) ширини.
Max-width допомагає визначити максимально можливу ширину об'єкта, натомість min-width дозволяє задати його мінімальну ширину.



Детальніше про це можна дізнатися за посиланням https://webdesignerwall.com/tutorials/5-useful-css-tricks-for-responsive-design

Більш детально з модулем **flexbox** можна ознайомитися за посиланням https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Вивчити властивості модуля **grid** допоможе гра «Морквяні городи» за посиланням https://codepip.com/games/grid-garden/

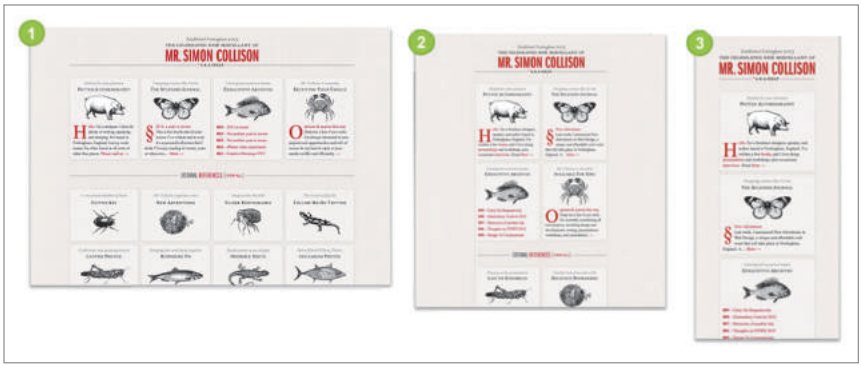


Рис. 11.58. Приклад адаптивної верстки на основі медіа-запитів

```
<div class="new-flex-container">
    <div class="new-flex-block">item1</div>
    <div class="new-flex-block">item2</div>
    <div class="new-flex-block">item3</div>
</div>
```

Рис. 11.59. Приклад оголошення **flexbox**

Flexbox дозволяє керувати елементами лише в одновимірному просторі, тому наступним кроком є розробка модуля grid, який дозволяє оперувати стовпцями та рядками.

Grid container (грід-контейнер) — це набір горизонтальних і вертикальних grid-ліній, що перетинаються. Ці лінії ділять простір на grid-області, де розташовуються grid-елементи. Усередині grid-контейнера є два набори grid-ліній: один визначає вісь стовпців, інший — вісь рядків.

Модуль grid також працює з елементом display (рис. 11.60).

```
<div class="grid">
        <div class="box item1">Item 1</div>
        <div class="box item2">Item 2</div>
        <div class="box item3">Item 3</div>
        <div class="box item4">Item 4</div>
</div>
```

```
.grid {
    display: grid;
    grid-gap: 20px;
    grid-templale: 100px auto 100px/1fr 80px 3fr 20%;
}
```

Рис. 11.60. Приклад використання модуля **grid**

Для тих, хто хоче досконало вивчити властивості flexbox, можна спробувати свої сили в грі Flexbox Froggy, де потрібно допомагати жабеняті Фрогі та його друзям написанням css-коду (https://flexboxfroggy.com/#uk).

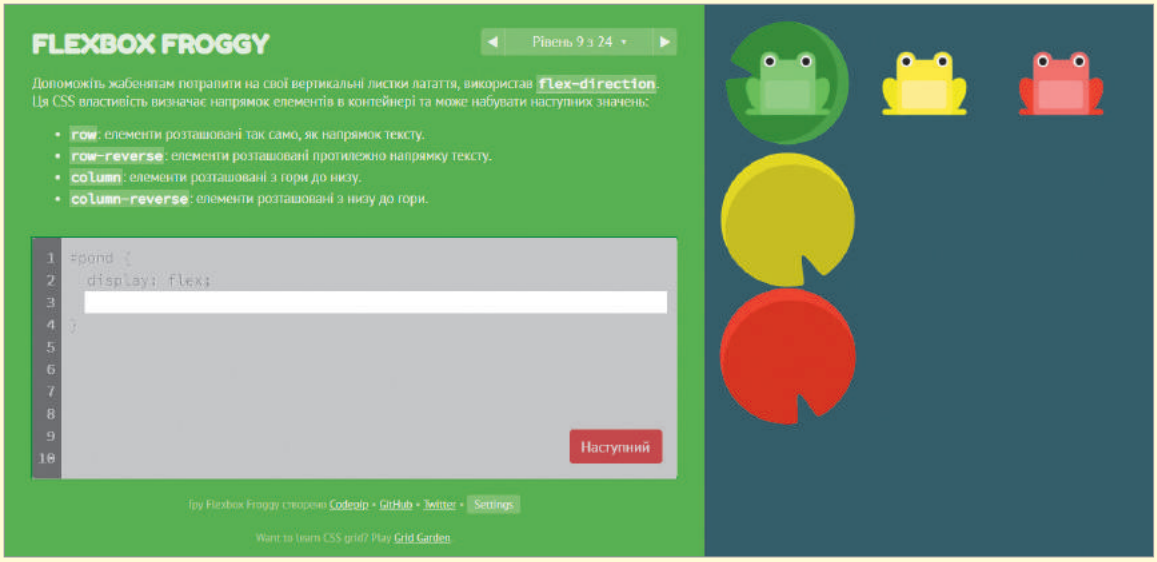


Рис. 11.61. Вікно гри **Flexbox Froggy**

Запитання для перевірки знань

1. Поясніть необхідність розробки адаптивних сайтів.
2. Що таке медіа-запити? Чим обумовлена їх поява?
3. Яка сфера використання медіа-запиту **braille**?
4. Опишіть принципи адаптивного дизайну.
5. Які інструменти дозволяють розробляти адаптивні сайти?
6. Поміркуйте, чому для адаптивної верстки краще використовувати векторну графіку.

Завдання для самостійного виконання

Для розуміння роботи адаптивного шаблону відкрийте сторінку Google.

1. Зменшуйте розмір вікна браузера і слідкуйте за тим, як змінюються положення блоків, зникають зображення, як перетворюється меню в результаті зменшення вікна браузера.
2. Опишіть зміни у відображенні сторінки під час застосування режиму інспекції.
3. Перегляньте код, наведений на рис. 11.62. З'ясуйте, який механізм використовують розробники сайта для досягнення адаптивності.

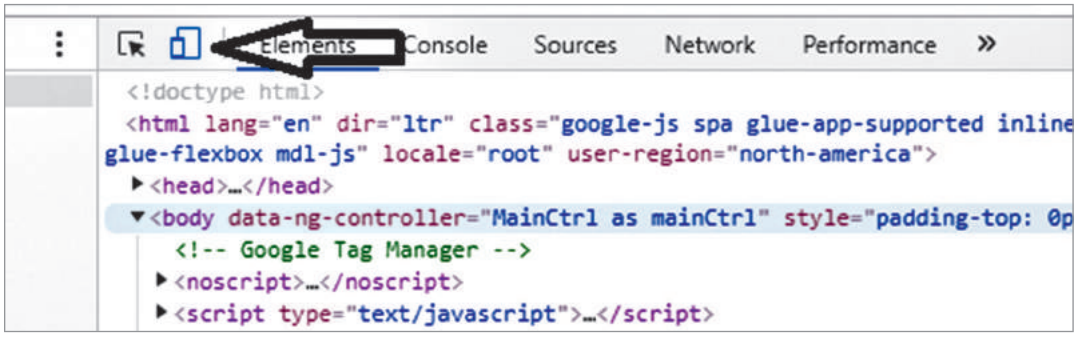


Рис. 11.62. До завдання 3

11.11. Кросбраузерність

Кожний користувач віддає перевагу певному браузеру. Хтось полюбляє вогняну лисичку, комусь до вподоби хром, а хтось не уявляє собі життя без сафарі. Поміркуйте, чи однаково обробляють браузери код сайта, який ви хочете переглянути?

Слово «браузер» (від англ. *Browser* — гортати) в англомовних програмах можна побачити на кнопках у діалогових вікнах. Крім браузера, найпопулярнішими варіантами перекладу є (веб-) перекладач або оглядач, а також (веб-) навігатор.

Найпопулярніші браузери
- Google Chrome
- Safari
- Mozilla Firefox
- Opera
- Internet Explorer

Як свідчить статистика, переважна більшість користувачів надає перевагу браузерам, які працюють на рушії Webkit, — Google Chrome і Safari (рис. 11.63).

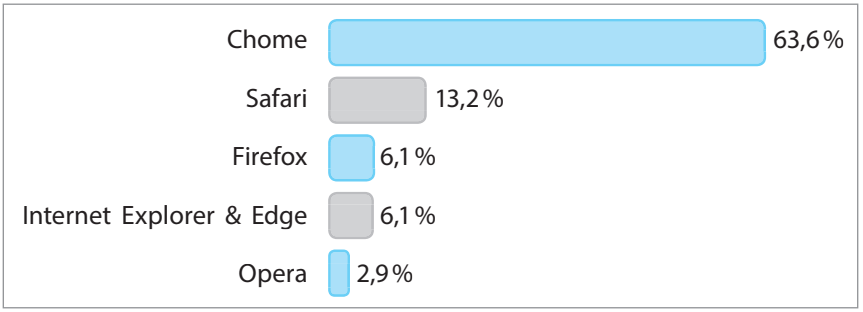


Рис. 11.63. Статистика використання браузерів

Кожен браузер має свою історію, свої версії, які, у свою чергу, розрізняються підтримкою Javascript, HTML і CSS. Хоча різні браузери в основному дотримуються загальних правил і стандартів. Проте в деяких випадках буває, що алгоритми обробки HTML-кодів і каскадних таблиць CSS можуть бути різними. Це призводить до різного відображення одного і того самого елемента сайта в різних браузерах.

Ознайомимося з історією створення популярних браузерів:

Рік	Опис
1994	З'явився один з перших вдалих браузерів Netscape Navigator, створений на основі першого браузера з графічною оболонкою NCSA Mosaic
1994–1996	Компанія Microsoft розробила браузер Internet Explorer, однак перші три версії широкого розповсюдження не отримали. Тривалий час браузери Netscape Navigator і Internet Explorer розвивалися паралельно, доки останній не захопив 95 % ринку. Далі в Internet Explorer застій (з 4.0 до 6.0 версії), а в Netscape Navigator, який був написаний на новому рушії Gecko, — відродження у версії 6.0. Оновлений Netscape Navigator не досяг колишніх вершин, та рушій Gecko у 2004 році послужив основою для створення сучасного браузера Mozilla Firefox однойменної компанії
1996	У компанії Opera Soft AS з'явилася Opera, швидка і проста у використанні
2003	Корпорація Apple випустила браузер Safari на рушії WebKit
2008	Корпорація Google випустила браузер Google Chrome на тому самому рушії, що й Safari

Розглянемо розвиток браузерів більш докладно.

Веб-браузер Тіма Бернерса Лі WorldWideWeb дозволяв переглядати текстові сторінки, перегляд зображень здійснювався в окремих вікнах і був з чорно-білим інтерфейсом. Перший

браузер, який отримав графічний інтерфейс, тобто не тільки просто текст на чорному тлі, був розроблений у 1993 році і мав назву NCSA Mosaic (рис. 11.64).

Рис. 11.64. Браузер **Mosaic**

Наступником Mosaic став браузер Netscape (рис. 11.65). Його розробники додавали в HTML нові теги, які робили зовнішній вигляд документа більш привабливим. Ці теги не були стандартизовані й працювали лише в Netscape. Та оскільки частка Netscape на той час становила понад 90 % від усіх наявних браузерів, це проблемою не було.

Поява конкурента — Internet Explorer від Microsoft — і практично відкрита війна між двома корпораціями за частку ринку, яка отримала назву «війна браузерів» (більш детально про війну браузерів можна дізнатись, переглянувши фільм «*Download: The True Story of the Internet*») призвела до серйозної проблеми, що полягала у відсутності єдиних стандартів відображення веб-сторінок. Дійшло до того, що на сайтах вбудовувалися кнопки Best viewed in Netscape і Best viewed in Internet Explorer.

Найбільші відмінності виникали у підтримці JavaScript — мови сценаріїв, що додає інтерактивності документам. У результаті багато документів було «оптимізовано» для конкретного браузера й абсолютно не читалися в іншому.

Сьогодні виробники браузерів почали активно займатися своєчасною підтримкою стандартів World Wibe Web, HTML5 і CSS3, що, безсумнівно, позначилося на якості відображення веб-сторінок. Проте у деяких властивостях каскадних таблиць і трактуванні коду навіть зараз на різних рушіях зустрічаються суттєві розбіжності, які необхідно виправляти вручну.

Рис. 11.65. Браузер **Netscape**

Рис. 11.66. Браузер **Mozilla**

Moziila — це внутрішнє ім'я браузера **Netscape Navigator**, що означає **Mosaic Killer** (вбивця **Mosaic**). Назва здалася співробітникам фірми занадто зухвалою, пізніше так було названо нащадка **Navigator** — **Mozilla Foudation.** Браузер же отримав назву **Phoenix** на честь птаха Фенікс, який згоряє, щоб відродитися з попелу. Згодом цю назву було змінено на **Firebird** (Жар-Птиця), а потім на **Firefox** (бо дві попередні назви вже використовувалися іншими розробниками).

Американський веб-дизайнер Ерік Майєр запропонував **CSS Reset** у травні 2007 року. Майєр написав цілу низку популярних у світі веб-розробки книжок, таких як «Cascading Style Sheets: The Definitive Guide», «Eric Meyer on CSS». Має власний сайт (https://meyerweb.com).

Вендорні префікси:

- -webkit — для Google Chrome, Safari і iOs
- -moz — для Mozilla
- -o — для Opera
- -ms — для Internet Explorer

Рис. 11.67. Плагін створення вендерних префексів

Кросбраузерність — це правильна верстка сайта, за допомогою якої веб-сторінки сайта однаково відображаються в різних браузерах. Реалізація відбувається за допомогою HTML і CSS, а також різноманітних хаків, в окремих випадках — JavaScript.

Для правильного відображення сайта одночасно в найбільш поширених браузерах, причому різноманітних версій (від найбільш ранніх до найновіших), веб-дизайнер обов'язково має дбати про кросбраузерність проекту, сайта з першої секунди роботи над ним.

Найбільш поширений спосіб, який застосовується багатьма веб-програмістами, — це написання так званих хаків — наборів спеціальних селекторів або правил, які розуміє тільки якийсь певний браузер. Тобто якщо необхідно коректно відображати сайт, скажімо, у трьох браузерах, то потрібно написати по хаку для кожного браузера. Є ще один спосіб — просто використовувати ті елементи при верстці html-коду, які у всіх необхідних браузерах відображаються однаково.

CSS-хаки — уривки коду, що розуміються тільки одним певним браузером. Хаки — найбільш «брудний» спосіб виправлення помилок, робить код не естетичним і не дійсним, але робочим.

У кожного браузера є свої вбудовані, експериментальні або нестандартні властивості, і для того щоб вони коректно працювали, використовують вендорні префікси. У назві недаремно використано слово префікси, бо, як і в граматиці, вони є приставкою, тільки в даному випадку до властивості CSS.

Вендорні префікси є ще одним рудиментом браузерних воєн, особливо браузерів WebKit, більш «чистий» і чесний спосіб, ніж використання хаків.

Кожен браузер має власні властивості з вендорним префіксом, так, наприклад, елемент border-radius у MozillaFirefox представлений властивістю -moz-border-radius, а в Chrome і Safari — -webkit-border-radius. Такі властивості змінюють поведінку елемента тільки в певному браузері та ігноруються іншими платформами.

Таким чином, вендорні префікси — приставки до стилів CSS, мають змістовне навантаження лише для тих браузерів, до яких належать. Вони дозволяють браузеру сприймати нестандартні властивості, а також стилі, призначені для інших користувальницьких клієнтів.

Можна скористатися плагіном Autoprefixer (https://autoprefixer.github.io) (рис. 11.67), який аналізує правила CSS, що існують на сайті, та додає необхідні вендорні префікси.

Як відомо, кожен браузер за замовчуванням має певний набір базових стилів, які він застосовує до сторінки. У різних браузерах ці правила трохи відрізняються. Щоб їх усунути та зробити за замовчуванням відображення сторінки у всіх браузерах однаковим, використовують спеціальні CSS-файли: reset.css або normalize.css.

Файл reset.css містить перелік усіх можливих HTML-тегів і скидає їх значення в нуль, тобто прибирає всі можливі відступи, робить шрифт однаковим у всіх тегах. Таким чином, всі заголовки й абзаци відображаються простим текстом, одним розміром і без відступів. Як наслідок відбувається скидання стилів за замовчуванням у всіх браузерах. Спочатку на сторінці користувачу слід підключити файл reset.css, а потім власний файл зі стилями style.css. У будь-якому браузері вся розмітка ґрунтуватиметься на тих стилях, які потрібно поставити в style.css.

Завантажити файл можна із сайта https://cssreset.com/

Рис. 11.68. Головна сторінка сайта **CSS-reset**

Скидати всі стилі не завжди доречно, саме тому існує ще один (дещо інший) інструмент — normalize. На відміну від normalize.css нормалізує стилі (тобто приводить до єдиного вигляду у всіх браузерах). Після його застосування базові стилі відображення заголовків, розмір шрифтів, відступи тощо уніфікуються і відображаються у всіх браузерах однаково.

Завантажити файл можна із сайта https://necolas.github.io/normalize.css

Потрібно віддавати перевагу універсальним елементам — тим, які однаково працюють у більшості браузерів. Робота лише з ними зробить код коротким, чистим і зрозумілим.

Наразі CSS-файли **reset.css** і **normalize.css** є найпопулярнішими й повністю відповідають HTML5. Найпростішим способом використання файлів є завантаження їх із сайтів у теку CSS та підключення на сторінках перед власним файлом стилів.

Normalize.css — продукт глибокого дослідження відмінностей між початковими стилями браузера. Дослідження провів Ніколас Галахер, взявши за мету зберігати корисні налаштування; нормалізувати стилі для більшості HTML-елементів; коригувати помилки й основні невідповідності браузера; удосконалювати юзабіліті непомітними поліпшеннями; пояснювати код, використовуючи коментарі та детальну документацію. Наразі normalize.css використовується в Twitter Bootstrap, HTML5 Boilerplate, GOV.UK, Rdio, CSS Tricks і в багатьох інших фреймворках, інструментах і сайтах.

Для досягнення правильного відображення сайта в різних браузерах слід: використати вендорні префікси; підключити CSS-файл normalize.css або reset.css; намагатися використовувати елементи, що мають однакове відображення в усіх браузерах.

Запитання для перевірки знань

1. Що таке кросбраузерність?
2. Які браузери ви знаєте?
3. Що таке CSS-хаки, який їх недолік?
4. Що таке вендорні префікси?
5. Яким чином досягається кросбраузерність сайта?
6. Знайдіть в Інтернеті відомості про «війну браузерів» та підготуйте невеличку презентацію.

11.12. Графіка для веб-середовища

Перші сайти містили виключно текстовий контент. Сучасні сайти наповнені різноманітними графічними об'єктами. Які, на вашу думку, інструменти необхідні для відображення графіки на сайті?

Михайла Комвуті-Веру (псевдонім Леа Веру) — запрошена експертка в W3C CSS Working Group, фахівчиня, визнана в галузі фронт-енду. Авторка книги «Таємниці CSS. Ідеальні рішення повсякденних задач», багатьох статей у популярному інтернет-журналі «A List Apart» та відомому блозі про можливості CSS3.

Фотографії, рисунки, фонові малюнки — все це візуальні елементи веб-дизайну. Саме від них залежить зовнішній вигляд сайта, а також швидкість його завантаження. Загалом у веб-дизайні використовують чотири основні формати графічних файлів, які вже знайомі вам з курсу 10 класу.

Формат JPEG (*Joint Photographic Experts Group*), або JPG, — це 16-бітовий формат растрових зображень, популярний для зберігання цифрових фотографій, які мають дрібні деталі та яскраві кольори. Завдяки тому, що зображення у форматі JPEG швидко завантажуються, його вибирає більшість веб-дизайнерів для своїх сайтів. Проте JPEG не підтримує прозорість, і його не має сенсу використовувати для логотипів і піктограм.

Формат GIF (*Graphic Interchange Format*) — це формат, обмежений 256 кольорами, дуже ефективний для зберігання логотипів, піктограм, таблиць. Проте GIF абсолютно не підходить для цифрових фотографій із тисячами відтінків. На відміну від формату JPG, він підтримує прозорість зображення та дозволяє створювати різноманітну анімацію (§ 11.13)

Формат PNG (*Portable Network Graphic*) — є, можна сказати, покращеним JPEG, і його зручно використовувати для простого й плоского графічного дизайну. PNG дозволяє працювати із прозорістю краще, ніж GIF, проте він не підтримує анімацію. Зазвичай цей формат використовують для публікації невеликих картинок, логотипів, іконок, діаграм, графічних елементів із прозорістю, фотографій без втрати якості.

Формат SVG (*Scalable Vector Graphics*) — найбільш поширений векторний формат, попит на який збільшується завдяки адаптивному дизайну у веб-розробці. Геометрична природа файлів SVG дозволяє легко адаптувати векторну графіку до потрібних параметрів. Розмір файла залежить від його складності, так, для простих зображень SVG-файл матиме менший розмір, ніж будь-який растровий аналог (JPEG, PNG). Формат ідеально підходить для зберігання значків, логотипів, діаграм. Серед його переваг — швидкість завантаження, чудове відображення на highDPI дисплеях, гнучкість і масштабованість тощо.

Зазначимо, що ми говоримо насамперед про статичні зображення. Їх легко зобразити, використовуючи елемент <img>.

Розглянемо тег <img> і його атрибути.

Елемент <img> є стандартним тегом для додавання графічних елементів на веб-сторінку. Цей тег вставляє зображення на сторінку в тому місці, де воно має з'явитися.

Все більше веб-дизайнерів пристають до думки, що растрові зображення повинні використовуватися лише для фотографій. За своєю суттю будь-який сайт — це інтерфейс, і всі інтерфейси мають бути векторними. Саме тому безумовну популярність набуває використання векторного формату SVG. Про особливості SVG та його використання можна дізнатися на сторінці http://yoksel.github.io/about-svg/?full#8

Тег <img> є непарним тегом.

- Обов'язковим атрибутом є src (англ. *source* — джерело), який містить шлях до зображення. Зазвичай зазначається або URL-адреса, або відносна адреса щодо місця розташування веб-сторінки, яка містить посилання на зображення.

З огляду на стандарти розробки сайтів (§ 11.9) рекомендується створювати окрему теку з назвою «images», в якій зберігаються всі зображення, що містяться на веб-сторінках сайта.

Тоді тег матиме такий вигляд:

```
<img src="images/2.jpg">
```

Тег відображає зображення лише графічних форматів GIF, JPEG, PNG і SVG. В атрибуті обов'язково має вказуватися розширення.

- Іншим атрибутом є alt (*alternative*), у якому вказується альтернативний текст — опис зображення для тих випадків, коли користувачі не можуть побачити картинку (приклад).

Атрибут alt має надати достатньо інформації користувачеві, щоб він склав уявлення про те, що є на зображенні:

```
<img src="URL" alt="альтернативний текст">
```

Нагадаємо, що в стандарті HTML5 теги мають лише семантичний зміст, а функції форматування покладено на каскадні таблиці стилів.

Розглянемо CSS-властивості зображень.

Розмір зображення задається двома параметрами: width — ширина зображення; height — висота зображення.

Якщо не задавати розміри зображення, то, по-перше, воно відобразиться на сторінці в реальному розмірі (рис. 11.69, *а*), по-друге — браузер потребуватиме часу на те, щоб дізнатися розміри і завантажити зображення. Лише після цього він повернеться до завантаження іншого вмісту документа й, таким чином, виведення решти елементів затримається. Крім того, при маленькому розмірі зображення й довгому альтернативному тексті, ще до того як завантажиться графіка, тимчасово відбудеться зсув дизайну сайта. Адже довгий альтернативний текст буде займати стільки місця, скільки йому знадобиться.

Якщо зазначити розміри зображення (рис. 11.69, *б*), то браузер спочатку зарезервує місце під зображення, підготує макет документа, відобразить текст і лише потім завантажить зображення.

Слід пам'ятати, що зображення можуть бути квадратними або прямокутними. Якщо ми використовуватимемо обидва параметри, браузер помістить зображення в прямокутник

Тег **<img>** використовують для розміщення фотографій, логотипів, графічних елементів інтерфейсу тощо.

Приклад. У випадку, коли користувачі мають вади зору, використовується скрін-рідер, який читає описи зображень.

BMP (*Bitmap Picture* — дослівно формат для зберігання растрових зображень) — один із найстаріших графічних форматів, який теж можна використовувати у веб. BMP розроблено компанією Microsoft і призначено для зберігання великих файлів усередині ОС Windows. Його особливість — не використовується жоден алгоритм стиснення, тому розмір файла зазвичай дуже великий.

виділеного розміру, навіть якщо його реальні ширина й висота більші (стисне) або менші (розтягне), як на рис. 11.69.

а б в

Рис. 11.69. Використання атрибутів розміру зображення:
у реальному розмірі (*а*); вказано ширину, браузер автоматично обчислив висоту (*б*); вказано однакові ширину і висоту, браузер відобразив прямокутну картинку як квадратну, тобто стиснув за шириною (*в*)

Тепер розглянемо приклад.

Приклад.

```
1 <img src="images/02.jpg" alt="Розміри зображення не вказані">
2 <img src="images/02.jpg" alt="Вказана ширина зображення" width="400">
3 <img src="images/02.jpg" alt="Вказані ширина та висота зображення" width="400" height="400">
```

Якщо ми задамо тільки один з атрибутів, то інший буде обчислюватися автоматично для збереження пропорцій малюнка.

Ширину й висоту зображення можна задавати як у пікселях (при цьому розмір картинки буде постійним незалежно від роздільності екрану), так і у відсотках — тоді розмір картинки залежатиме від роздільності екрана користувача.

Параметр border дозволяє створити рамку для зображення. Причому можна налаштовувати ширину та колір і стиль рамки за допомогою відповідних параметрів:

- border-width;
- border-color;
- border-style.

Додатково можна налаштовувати ці властивості для кожної сторони:

- top (верхня),
- left (ліва),
- right (права),
- bottom (нижня).

Якщо вказати {margin: 50px;}, ми задамо відступи в усіх напрямках: зверху, знизу, зліва і справа. Задати відставання для різних напрямків можна за допомогою скороченого оголошення.
Наприклад,

- margin: 10px 20px 50px 100px;
- еквівалентно запису
- margin-top: 10px;
- margin-right: 20px;
- margin-bottom: 30px;
- margin-left: 40px;

Наприклад, border-left-width, border-bottom-color, border-right-style.

- padding — внутрішні відступи між зображенням та рамкою;
- margin — задає відступи від зображення від усіх чотирьох країв;

Можна задавати відступ від конкретного краю (рис. 11.70). Наприклад,

- margin-top — відступ від верхньої сторони;
- margin-right — відступ від правого боку;
- margin-bottom — відступ від нижньої сторони;
- margin-left — відступ від лівого боку.

Аналогічно можна задавати внутрішні відступи: padding-top, padding-left, padding-right, padding-bottom.

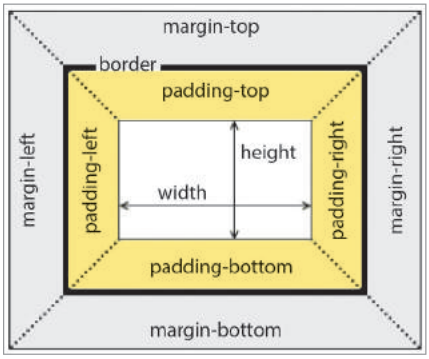


Рис. 11.70. Відступи в CSS

Запитання для перевірки знань

1. Які графічні формати та якими командами мови розмітки відображаються?
2. Що таке screen-reader? Для чого він призначений?
3. Які обов'язкові атрибути має тег для відображення графічних об'єктів на веб-сторінці?
4. Які стилі використовують для відображення графічного об'єкта на веб-сторінці?
5. Обґрунтуйте використання атрибутів ширини та висоти зображення.
6. Який вид комп'ютерної графіки краще використовувати для фотографій?

Завдання для самостійного виконання

Створіть сторінки, на яких відображатимуться графічні об'єкти, аналогічно рис. 11.71. Результат надішліть на електронну пошту вчителю.

а б в

Рис. 11.71. До завдання

11.13. Анімаційні ефекти

Пригадайте, як часто ви бачите сайти, на яких немає жодної анімації.

Анімація є одним із трендів у дизайні веб-інтерфейсів, вона вже давно стала невід'ємною частиною кожного сайта. Анімаційні ефекти можна застосовувати, як до окремих об'єктів, так і до зображень.

Сучасні можливості анімації дозволяють зробити найрізноманітніші слайдери: на повний екран, з 3D-ефектами, адаптивні (які переглядаються з будь-яких мобільних пристроїв) та ін.

Анімація — це і крихітні, ледь помітні індикатори завантаження, і цілі сторінки, на яких вам наче показують фільм.

Важко назвати галузь веб-дизайну, де б не використовувались анімаційні ефекти: від декоративних елементів, що просто прикрашають інтерфейс, до ефектів, які активно впливають на користувача.

Перший варіант (найпростіший), який ми розглянемо, — *анімовані об'єкти*. Це окремі зображення і об'єкти, які показуються користувачеві. Наприклад, курсор, що рухається, плаваюча кнопка «вгору», кнопки заклику до дії, зміна кольору тощо.

Другий варіант — *анімовані зображення* (наприклад, рекламні банери на сайті). Вони можуть вести на внутрішні сторінки, блог, перенаправляти відвідувача на інший сайт. Ще один приклад — слайд-шоу з фотографій.

Анімаційні ефекти можна створювати як виключно засобами каскадних таблиць стилів, так і за допомогою JavaScript.

CSS-анімація робить можливим анімацію переходів (transitions) з однієї конфігурації CSS-стилю до іншої. Анімація складається з двох компонентів, а саме: зі стилю, котрий описує CSS-анімацію, та набору ключових кадрів (keyframes), які задають початковий і кінцевий стани стилю анімації (також є можливість задання точок проміжного стану).

Розглянемо алгоритм створення CSS-анімації.

Основні переваги CSS-анімація перед традиційними скриптовими техніками анімації:

- легка у використанні для простих анімацій та не потребує знання **JavaScript**;
- чудово функціонує навіть під час помірного навантаження на систему (на відміну від анімації **JavaScript**).

Крок 1	Починається з оголошення імені анімації в блоці @keyframes і визначення так званих кроків анімації, або ключових кадрів.
Крок 2	Після оголошення відкривається фігурна дужка (у нашому прикладі на чистому CSS), у якій послідовно від 0 до 100 % прописуються властивості для кожного ключового кадру. Між цими значеннями можна вставляти скільки завгодно проміжних значень, наприлад 50 %, 75 % або навіть 83 %.
Крок 3	Наступною командою є використання CSS властивості animation.

Приклад простої анімації — Плавне мерехтіння (зміна прозорості) наведено на рис. 11.72.

Розглянемо спочатку таблицю властивостей animation, а потім приклад.

```
.element{
    animation: simpleAnimation 5s infinite;
}
@keyframes simpleAnimation {
    0%{
        opacity: 0;
    }
    25%{
        opacity: 0.4;
    }
    50%{
        opacity: 0.8;
    }
    75%{
        opacity: 0.4;
    }
    100%{
        opacity: 0;
    }
}
```

Рис. 11.72. Код простої анімації

Властивість **opacity** дозволяє зробити будь-який елемент веб-сторінки частково або повністю прозорим. **Opacity** змінює прозорість елементів, для яких встановлено фонове зображення або задано тло за допомогою кольору або градієнта. Якщо елементи містить інші елементи, вони також змінять прозорість. **Opacity** набуває значення в діапазоні від 0 (повністю прозорий) до 1 (непрозорий).

Анімація	Опис
animation-delay	Змінює час затримки між часом із моменту завантаження елемента та початком анімаційної послідовності
animation-direction	Визначає зміну напрямку анімації та його чергування залежно від кількості проходів анімації, може задавати повернення в початковий стан і починати прохід заново
animation-duration	Визначає тривалість циклу анімації
animation-iteration-count	Визначає кількість проходів (повторів) анімації; можна також обрати значення infinite для нескінченного повтору анімації
animation-name	Задає ім'я для анімації @keyframes через at-правило, яке описує анімаційні ключові кадри
animation-play-state	Дозволяє призупиняти й відновлювати анімацію
animation-timing-function	Задає конфігурацію таймінгу анімації; інакше кажучи, як саме анімація робитиме прохід через ключові кадри, це можливо завдяки кривим прискорення
animation-fill-mode	Визначає, які значення будуть застосовані для анімації перед початком і після її закінчення

Приклад.

Зазвичай розробники не пишуть про всі ці властивості окремо, а використовують короткий запис такої структури:

animation: (1. animation-name — назва)
(2. animation-duration — тривалість)
(3. animation-timing-function динаміка руху)
(4. animation-delay — пауза перед стартом)
(5. animation-iteration-count — кількість виконань)
(6. animation-direction — напрямок).

Animation-iteration-count може набувати такого значення: будь-яке додатнє число, що вказує кількість повторень, або infinite — нескінченну кількість разів, як у випадку нашого прикладу (рис. 11.72).

У нашому прикладі Звичайна анімація 5 секунд НЕСКІНЧЕННА.

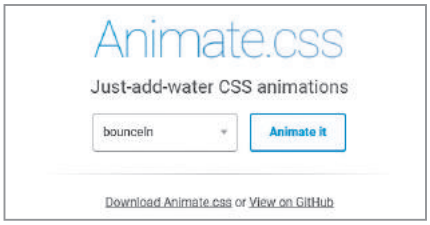


Рис. 11.73. Головна сторінка Animate.css

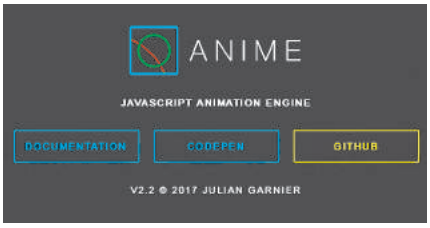


Рис. 11.74. Головна сторінка **Anime.js**

Крім того, існує не менше сотні різноманітних плагінів та бібліотек. Розглянемо лише декілька з них.

Animate.css — це фундаментальна бібліотека анімацій, сумісних з усіма браузерами та відповідних для безлічі завдань (рис. 11.73).

Animate.css містить все — від класичних підскакувань до останніх новинок і унікальних ефектів — і здатна задовольнити потреби практично будь-якого проекту.

Anime.js — це вражаючий набір функцій, які дозволяють пов'язувати безліч анімацій, синхронізувати етапи, малювати лінії, змінювати форму об'єктів, створювати власні анімації тощо (рис. 11.74).

CSS-Animate — це простий майданчик для написання робочого коду для будь-якої анімації.

Достатньо задати ім'я, клас, властивості анімації і фрейму, і можна керувати часовою послідовністю і додавати маркери.

Інакше кажучи, потрібно налаштувати все необхідне для створення стандартної анімації, заснованої на ключовому кадрі, як зображено на рис. 11.75.

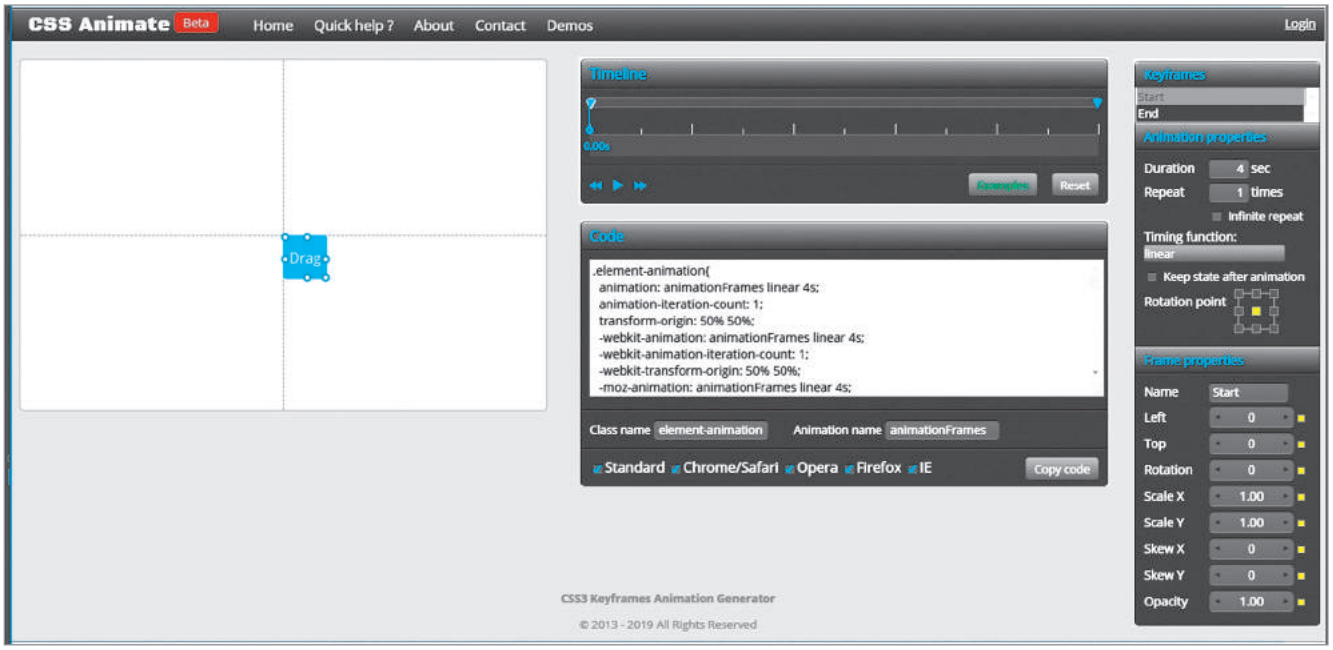


Рис. 11.75. Головна сторінка **CSS-Animate**

Запитання для перевірки знань

1. Наведіть приклади анімації на веб-сторінці.
2. Якими засобами можна створювати анімації?
3. Опишіть послідовність команд у CSS для створення анімації.
4. Перегляньте відео на **YouTube**. Спробуйте створити просту анімацію за інструкцією.
5. Дослідіть запропоновані плагіни, створіть засобами табличного процесора порівняльну таблицю характеристик.

11.14. Мультимедіа на веб-сторінках

Коли ми «в один клік» створюємо власний відеоканал на YouTube, викладаємо фотозвіт про нашу мандрівку в «живий журнал», коментуємо нову книгу у своєму блозі — інакше кажучи, робимо звичні нам речі, ми навіть не згадуємо про технології, які дають змогу робити це так легко й невимушено.

Технології опрацювання мультимедіа зараз є одним із найперспективніших і найпопулярніших напрямків сучасної інформатики. Мета — створення продукту, який передає інформацію шляхом упровадження та використання нових технологій, набору зображень, текстів і даних, що супроводжуються звуком, відео, анімацією й іншими візуальними ефектами.

Мультимедіа — комп'ютеризована технологія, що об'єднує роботу зі всіма джерелами даних, засіб подання різних видів інформації у цифровому вигляді.

Ознайомимося з тим, які об'єкти належать мультимедіа (рис. 11.76). Ви вже знаєте, як вставляти графічні й текстові об'єкти у веб-сторінку. Далі розглянемо, як працювати з аудіо- та відеооб'єктами.

Слід зазначити, що всі дані, надіслані мережею, позначаються певними назвами, які однозначно вказують їх тип, — так званими MIME (*Multipurpose Internet Mail Extensions*, багатоцільові розширення пошти Інтернету).

Тип MIME привласнює даним та сама програма, що їх і надсилає, наприклад веб-сервер (приклад). А браузер (тобто програма, яка їх отримує) визначає за типом MIME цих даних, чи підтримувати цей тип даних, і якщо так, що саме з ними робити.

Звукові файли мають розширення .wav, .au, .aif та ін. Фільми у форматі QuickTime мають розширення .qt або .mov, відео від Microsoft (*Microsoft Video for Windows*) — .avi (*Audio Video Interface*). Відеофайли (.mpg або .mpeg) у форматі MPEG зазвичай мають великий розмір, забезпечують високу якість відео, формат MPEG-4 дуже часто використовується при програванні відеофайлів online.

У таблиці наведено типи MIME-форматів мультимедійних файлів, які підтримуються браузерами:

Рис 11.76. Класифікація об'єктів мультимедіа

Приклад.

- Веб-сторінка має тип MIME text / html.
- Графічне зображення формату GIF має тип MIME image / gif.
- Свої типи MIME мають і мультимедійні файли.

Сучасні браузери працюють з кількома форматами мультимедійних файлів із десятків наявних сьогодні. Не слід забувати, що різні браузери підтримують різні формати файлів.

Формат файлів	Тип MIME	Браузери, якими підтримується
MPEG4	video/mp4	Chrome, Safari (не підтримується Firefox, Opera)
OGG, OGA, OGV	audio/ogg (для аудіофайлів) video/ogg (для відеофайлів)	Chrome, Firefox, Opera
MP4	video/mp4	Chrome, Safari
WEBM	video/webm	Chrome, Firefox, Opera

Тег **<embed>** використовується досить рідко. Його призначення — вбудовування об'єктів на веб-сторінку, наприклад, флеш-ролики або флеш-ігри.

Відео у форматі AVI на сайті засобами HTML5 не відтворюється. Його слід конвертувати у формати, які підтримуються браузерами. Можна скористатися онлайн-конверторами на кшталт VIDEO-CONVERTER (https://convert-video-online.com/).

HTML5-відео — стандарт для розміщення мультимедійних файлів у мережі з оригінальним програмним інтерфейсом без залучення додаткових модулів.

За допомогою елемента <video> з'явилася можливість додавати відеовміст на веб-сторінки, а також стилізувати зовнішній вигляд відеоплеєра за допомогою css-стилів.

Для відображення на сторінці відеозапису необхідно використовувати тег <video>, у якому використовується тег <source>, що має обов'язковий атрибут src, який визначає адресу відео.

Атрибут	Опис
Autoplay	Автоматичне відтворення відеофайла відразу після завантаження сторінки
controls	Вказівка браузеру, що потрібно відобразити базові елементи управління відтворенням (відтворення, пауза, гучність)
height	Задання висоти вікна для відображення відеоданих, можливі значення: px або %
loop	Циклічне відтворення відеофайла
muted	Вимикання звуку під час відтворення відеозапису
poster	URL-файл зображення, яке відображатиметься під час завантаження відеофайла або доти, поки користувач не натисне на кнопку PLAY. Якщо атрибут не задано, буде відображатися перший кадр відеофайла
preload	Атрибут, який відповідає за попереднє завантаження відеоконтенту. Не є обов'язковим, деякі браузери ігнорують його
auto-	Браузер завантажує відеофайл повністю, щоб він був доступний, коли почнеться відтворення
metadata	Браузер завантажує першу невелику частину відеофайла, щоб визначити його основні характеристики
none-	Відсутність автоматичного завантаження відеофайла
src	Містить абсолютну або відносну URL-адресу відеофайла
width	Задає ширину вікна для відображення відеоданих, можливі значення: px або %

Згадаємо, що перед використанням HTML5 елементів необхідно покроково зробити такі дії.

Крок 1	Указати правильний доктайп: <!DOCTYPE html>

Крок 2	У стилях CSS позначаємо тип елемента HTML <video> як блочний video
	{
	display: block;
	}

Крок 3

Для досягнення кросбраузерності доцільно перераховувати в <source> усі формати, починаючи з більш пріоритетного, та зазначати тип MIME для кожного відеофайла.
З огляду на стандарти розробки сайтів (для дотримання правил створення структури сайта), варто в теці Сайт створити додаткову теку video (audio), де розмістити відконвертовані відео- чи аудіофайли.
В елементі <video> краще використовувати атрибут controls, який відповідає за відображення елементів керування плеєром.

```
<video controls>
 <source src="video/movie.mp4" type="video/mp4">
 <source src="video/movie.webm" type="video/webm">
 <source src="video/movie.ogv" type="video/ogg>
</video>
```

До появи стандарту HTML5-відео використовувався елемент <embed>, який визначає контейнер для зовнішнього застосування або інтерактивного вмісту (іншими словами, плагіна).

Більшість браузерів протягом довгого часу підтримувало цей елемент. Проте даний тег не був включений у специфікацію HTML4, його додали в специфікацію HTML5.

Наразі SWF — це основний формат для показу анімованої векторної графіки у вебі.

Запитання для перевірки знань

1. Дайте означення мультимедіа.
2. Наведіть приклади об'єктів мультимедіа.
3. Що таке формат MIME?
4. Поясніть необхідність конвертації відеофайлів.
5. Які теги для яких форматів використовують?

Завдання для самостійного виконання

1. Додайте на першу сторінку завдання § 11.12 два відеофайли (можна однакові), щоб вони відображалися, як наведено на рис. 11.77. Результат надішліть учителю.

   Підказка: використовуйте контейнер зі стилем {display: inline-block;}

Рис. 11.77. Відображення відеофайлів

2. Додайте на другу сторінку завдання § 11.12 після графічного об'єкта аудіофайл. Результат надішліть учителю.
3. Використовуючи тег <embed>, додайте будь-який флеш-ролик на третю сторінку завдання § 11.12 (як допомогу перегляньте відео https://www.youtube.com/watch?v=i_l4X2K6PbE).

11.15. Об'єктна модель документа

Коли користувач уводить в адресний рядок назву сайта, браузер завантажує HTML-сторінку, після чого створює об'єктну модель документа — DOM.

Об'єктна модель документа (англ. *Document Object Model*, DOM) — це програмний інтерфейс (API) для HTML.

DOM є стандартом, запропонованим веб-консорціумом **W3C** (*World Wide Web Consortium* — консорціум Всесвітньої павутини), і регламентує спосіб подання вмісту документа (зокрема веб-сторінки) у вигляді набору об'єктів.

DOM надає структуроване уявлення про документ та уможливлює доступ до цієї структури програмам, які можуть змінювати вміст, стиль і структуру документа. Подання DOM складається із структурованої групи вузлів і об'єктів, які мають властивості і методи. Власне, DOM з'єднує веб-сторінку з мовами опису сценаріїв або мовами програмування.

Усі властивості, методи і події, доступні для керування й створення нових сторінок, організовані у вигляді об'єктів.

Веб-сторінка — це документ, який може бути поданий як у вікні браузера, так і в самому HTML-коді. У будь-якому випадку, це один і той самий документ. DOM надає інший спосіб подання, зберігання й керування цього документа. Він повністю підтримує об'єктно орієнтоване уявлення веб-сторінки, роблячи можливим її зміну за допомогою мови опису сценаріїв на кшталт JavaScript.

DOM подає HTML-теги у вигляді об'єктів із властивостями і методами. У кожного HTML-тега (об'єкта) на HTML-сторінці, завдяки DOM, є своя унікальна адреса. Отримуючи доступ за цією адресою, JavaScript може керувати HTML-тегом.

Відкриваючи HTML-сторінку, браузер створює на основі її тегів структуру DOM, де кожен HTML-тег постає у вигляді об'єкта зі своєю унікальною адресою. Після аналізу структурованого документа будується його подання у вигляді дерева (рис. 11.78, 11.79).

```
<html>
  <head>
    <title>My title</title>
  </head>

  <body>
    <h1>My header</h1>
    <a href="#">My link</a>
  </body>
</html>
```

Рис. 11.78. DOM

Дерево в моделі DOM складається із множини зв'язних вузлів (Node) різних типів. Усе, що є в HTML, знаходиться і в DOM. Навіть директива <! DOCTYPE ...>, яку ми ставимо на початку HTML, теж є DOM-вузлом і знаходиться в дереві DOM безпосередньо перед <html>.

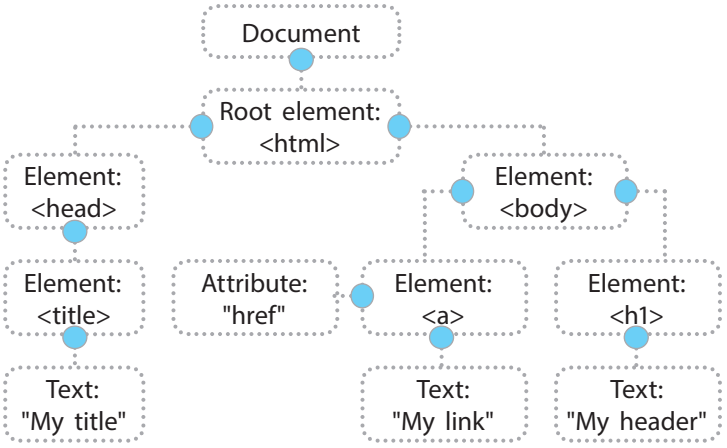


Рис. 11.79. Дерево вузлів DOM

DOM не є мовою програмування, але без нього JavaScript не мав би жодної моделі або уявлення про веб-сторінку, HTML-документ, його елементи. Спочатку JavaScript і DOM були тісно пов'язані, але згодом розвинулися в різні сутності. Вміст сторінки зберігається в DOM і може бути доступним і змінюватися з використанням JavaScript.

Таким чином, можна сказати, що DOM — це веб-технологія, що дозволяє керувати HTML-тегами сторінки через мову JavaScript.

Зазвичай розрізняють вузли декількох типів:

Вузол	Опис
Документ (Document)	Корінь дерева, представляє цілий документ, інакше кажучи — точка входу в DOM
Фрагмент документа (DocumentFragment)	Вузол, який є коренем піддерева основного документа
Елемент (Element)	Представляє окремий елемент HTML, можна сказати, що це основні будівельні блоки. Атрибут (Attr) представляє атрибут елемента
Текст (Text)	Представляє текстові дані, які містяться в елементі або атрибуті
Коментарі	Іноді можна включити інформацію, яка не буде показана, проте є доступною з JS

Стандартом визначаються деякі інші типи вузлів у моделі документа. Вузли деяких типів можуть мати гілки, інші можуть бути лише листям дерева.

Створення DOM-елементів:

- document.create Element (tag); — створює новий тег.
- document.create TextNode (text); — створює текстовий вузол.

Додавання DOM-елемента:

- parentElem.appendChild (elem); — додає елемент у кінець дочірніх елементів;
- parentElem.insertBefore (elem, nextSibling); — вказує, перед яким із дочірніх елементів додати новий вузол.

Видалення DOM-елементів:

- parentElem.removeChild (elem) — видаляє конкретний елемент зі списку дітей батьківського елемента;
- parentElem.replaceChild(newElem, elem) — видаляє вказаний елемент і заміняє його новим.

DOM — стандарт, що розвивається і поділяється на три рівні. Перший рівень є першою версією стандарту і поки що єдиною закінченою. Він складається з двох розділів: перший є ядром і визначає принципи маніпуляції зі структурою документа (генерація і навігація), а другий присвячений поданням у DOM елементів HTML, що визначаються однойменними тегами. Другий і третій рівні описують модель подій, доповнюють таблиці стилів, проходи по структурі.

Запитання для перевірки знань

1. Що таке об'єктна модель документа?
2. Яка організація і чому запропонувала стандарт DOM?
3. Як можна представити веб-сторінку?
4. У чому перевага використання DOM?
5. Назвіть типи вузлів.
6. Як саме можна працювати з веб-документом завдяки DOM?

11.16. Веб-програмування та інтерактивні сторінки

Ви вивчаєте основи програмування. Поміркуйте, чи можна використовувати мови, які ви вивчаєте, для розробки сайтів?

Брендан Айк — американський програміст, автор мови програмування JavaScript, що є стандартом для веб-програмування. Він брав участь у створенні американської компанії Mozilla, яка розробляє браузер Firefox.

Історія створення JavaScript починається в 1995 році, у самий розпал війни Netskape і Microsoft (див. про «першу війна браузерів» у § 11.11). Це був час, коли анімації, взаємодія з користувачами та інші види інтерактивності мали стати невід'ємною частиною Інтернету майбутнього.

Веб потребував легкої скриптової мови (мови сценаріїв), спроможної працювати з DOM (§ 11.15), який ще не було стандартизовано. Така мова, як пригадує її розробник Брендан Айк, була створена за 10 днів. Вона подавалась як скриптова для виконання невеликих клієнтських завдань у браузері.

Веб-програмування — галузь веб-розробки і різновид дизайну, в завдання якої входить проектування користувальницьких веб-інтерфейсів для сайтів або веб-додатків.

JavaScript — мова програмування, що дозволяє реалізувати низку складних рішень у веб-документах. Вона допомагає зробити сторінки сайта більш інтерактивними, обробляє дії користувачів сайта. Це об'єктно-орієнтована клієнтська мова, яка підтримується додатками, що працюють з дизайном сайта.

Разом з HTML і CSS JavaScript — третій важливий блок, на основі якого будується більшість стандартних веб-інтерфейсів

Функції JavaScript дозволяють:

- зберігати дані в змінних
- активувати частину коду згідно з певними сценаріями, що реалізуються на сторінці сайта
- створювати контент, який оновлюється автоматично
- керувати мультимедійними можливостями (працювати з відео, анімувати зображення)

JavaScript обробляється у веб-додатках на стороні клієнта, тобто у браузері. Завдяки цьому він може виконуватися на будь-якій операційній системі, а веб-інтерфейси, що працюють на його основі, є кросплатформними.

Мова JavaScript має широке застосування в таких областях.

AJAX (англ. *Asynchronous Java Script And XML* — асинхронний JavaScript (мова програмування) і XML (мова розмітки веб-сторінок)) — дає змогу створювати набагато зручніші веб-інтерфейси користувача на тих сторінках сайтів, де необхідна активна взаємодія.

Поки сервер обробляє запит, користувач може переглядати вміст сайта. Браузер лише довантажує потрібні йому дані. Рівень використання AJAX значно підвищився після того, як компанія Google почала активно використовувати його у створенні своїх сайтів, таких як Gmail, GoogleMaps і GoogleSuggest.

Синтаксис **JS** навмисно було розроблено максимально подібним до **Java**, який на той момент дуже активно використовувався, а динамічну типізацію запозичено від не менш популярної мови **Perl**. Функції в **JavaScript** — це просто ще один тип об'єкта. Ними можна оперувати, як і будь-якими іншими елементами. Цією особливістю **JavaScript** зобов'язаний **Scheme**. Наслідування реалізовано через прототипи, як у мові **Self**.

Прикладами застосування технології AJAX є відображення контенту, що періодично оновлюється (інтерактивні карти); створення анімації й графічних об'єктів у форматі 2D/3D.

Поява формату JSON (англ. *JavaScript Object Notation 0151* — запис об'єктів JavaScript) стала наступним важливим кроком. JSON був розроблений у 2001 році Дугласом Крокфордом. Цей легкий формат, який використовується для обміну даними, заснований на підмножині мови JavaScript (спосіб створення об'єктів у JavaScript), яка використовувала можливості звичайного браузера та дозволяла веб-розробникам створювати веб-додатки з постійним двостороннім зв'язком із веб-сервером.

Найкраще JSON працює разом із AJAX. Джеймс Гаррет запропонував цей формат у 2005 році. Пригадаємо, що суттєвою перевагою цього формату є можливість довантажувати дані, не перевантажуючи сторінку (про JSON і AJAX далі, у § 11.21).

Comet — спосіб роботи веб-додатків, коли під час HTTP-з'єднання сервер відправляє дані браузеру без додаткових запитів.

Браузерні ОС — код деяких браузерних операційних систем, який складається переважно (іноді понад 75 %) зі скриптів.

Закладки — JavaScript має широке застосування в роботі програми, що розміщуються в закладках браузера.

Браузерні скрипти — програмні модулі, які пишуться на цій мові і дають дуже багато можливостей (автозаповнення форм, зміна формату сторінок, приховування небажаного змісту, додавання інтерактивних елементів на сторінках).

Серверні додатки — фрагменти коду, які виконуються на стороні сервера, де використовується Java 6.

Мобільні додатки — JavaScript може бути корисною в цьому популярному напрямку.

Віджети — на мові JavaScript пишуться різні міні-програми, які використовуються в робочому просторі і є дуже зручними.

Прикладне програмне забезпечення — об'єктно-орієнтована мова JavaScript використовується для створення окремих програм, у тому числі нескладних ігор. Наразі JavaScript є однією з найбільш популярних клієнтських мов.

Почувши словосполучення «інтерактивна сторінка», багато хто асоціює його з різними flash-ефектами, що активуються залежно від місця розташування курсору миші відвідувачів сайта. Проте цей стереотип дещо неправильний. Для реалізації інтерактивних «властивостей» сайтів застосовуються спеціальні програмні коди — серверні скрипти. Саме вони обробляють дані, отримані від відвідувачів сайта, і формують відповідну html-сторінку.

Для написання серверних скриптів застосовуються серверні мови веб-програмування, такі як PHP, Perl, ASP.NET. Виконується серверний скрипт на стороні сервера: відвідувач не бачить вихідного програмного коду виконуваного скрипта, а отримує тільки готову відповідь.

Створити інтерактивну сторінку — означає створити сторінку, що вміє «спілкуватися» зі своїми відвідувачами. Проста статична сторінка доступна лише для перегляду. Для того

Дуглас Крокфорд (на фото зліва) — американський програміст, творець текстового формату обміну даними **JSON**.

Джеймс Гаррет — основоположник науки «інформаційна архітектура», засновник компанії **Adaptive Path**, яка займається консультаціями зі створення сайтів, дружніх до користувачів.

Переваги JavaScript:

- підтримує всі браузери;
- перевіряє реєстраційні форми на помилки ще до відправлення на сервер;
- створює яскраві й інтерактивні сторінки сайта;
- може здійснювати різного типу обчислення.

щоб зв'язатися з адміністрацією сайта або зробити замовлення, відвідувачеві необхідно зателефонувати за контактним телефонним номером, написати електронного листа на e-mail або відправити факс. Інтерактивність сторінки насамперед забезпечується за допомогою HTML-форм: реєстраційних, відправки повідомлень, онлайн-замовлень тощо.

Під інтерактивним слід розуміти сайт, контент якого формується «на льоту». Поняття «інтерактивний» можна віднести до всіх сайтів, що мають форми відправки повідомлень, онлайн-анкети та опитування, реєстраційні форми, лічильники відвідувань, форми для онлайн-замовлень та інші подібні елементи.

HTML-форми — це елементи управління для збирання інформації від відвідувачів веб-сайта. Такі форми є набором текстових полів, кнопок, списків та інших елементів управління, які активізуються клацанням миші. Завдання форми полягає в передаванні даних користувача віддаленому серверу.

Для отримання й опрацювання таких даних використовуються мови веб-програмування: PHP або Perl.

У HTML5 розроблено дуже потужний інструментарій для створення форм будь-якої складності. Далі ми розглядатимемо лише створення найпростіших форм.

Базою для форми є парний тег <form> — контейнер, який утримує всі елементи керування форми, — поля, що групуються залежно від призначення форми. Загалом можна сказати, що кожна форма є набором логічно пов'язаних елементів.

За допомогою CSS-стилів можна змінювати розмір, тип і колір шрифта, інші властивості тексту, а також додавати межі, колір тла і зображення тла. Ширина поля задається властивістю width.

Серед атрибутів тегу є два обов'язкові: action і method.

Action містить url-адресу, яка визначає, куди буде надіслано дані форми.

Method визначає спосіб відправлення даних із форми. Він може набувати двох значень: "get" (за замовчуванням надсилає дані на сервер через адресний рядок, тобто їх видно в адресному рядку браузера) та "post" (приховує запит). Зазвичай використовується "post", оскільки він дозволяє передавати великі обсяги даних.

Елемент <input> є одинарним (непарним) тегом і створює більшість полів форми. Атрибути елемента відрізняються залежно від типу поля, для створення якого використовується цей елемент, тому першим обов'язковим його атрибутом є <type>.

Найчастіше використовувані значення атрибута type наведено в таблиці:

Назва	Опис
Text	Створює у формі текстові поля, виводячи однорядкове текстове поле для введення тексту
Submit	Створює стандартну кнопку, що активізується клацанням миші, збирає інформацію з форми й надсилає серверу для опрацювання
Reset	Кнопка для повернення даних форми в початкове значення
Placeholder	Містить текст, який відображається в полі введення до заповнення (найчастіше це підказка), тобто в текстовому полі відображатиметься текст підказки, який зникне, щойно буде введено певний текст
E-mail	Браузери, що підтримують цей атрибут, очікують, що користувач уведе дані, відповідні синтаксису адреси електронної пошти
Password	Створює у формі текстові поля, при цьому символи, що вводяться користувачем, замінюються на зірочки (або інші значки, встановлені браузером)

Слід наголосити, що type = "email" і type = "password" забезпечують специфічну семантику для введення, визначаючи, яку інформацію повинно містити поле.

Другий атрибут, який необхідно зазначати, це <name> — ім'я поля, необхідне для правильного опрацювання даних на сервері. Зазвичай воно повинне бути унікальним, принаймні в межах форми, для його імені використовується латиниця.

Для оформлення форми незайвим буде робити підписи до полів. Для цього існує спеціальний тег — парний тег <label>. Його завдання — створення логічного зв'язку між текстом і полем введення. Крім того, якщо клацнути на текст такого підпису, то курсор переміститься у відповідне поле.

Найпростіший спосіб створити підпис — це просто обернути текст підпису і тег поля на тег <label> таким чином:

```
<label>
Введіть ім'я <input type = "text" name = "username">
</label>
```

Більш детально ознайомитися з можливостями HTML-форм можна за посиланням https://css.in.ua/html/tag/form, https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form

Запитання для перевірки знань

1. Що таке веб-програмування?
2. Яка мова програмування є базовою у веб-розробці?
3. Наведіть приклади використання JavaScript.
4. Опишіть призначення форм. З яких елементів складається форма?
5. Як досягається інтерактивність сторінок?
6. Розгляньте такі елементи форми: текстові поля введення (<textarea>), розкривні списки (<select>), кнопки (<button>), прапорці (<input type = "checkbox">), перемикачі (<input type = "radio">). Інформацію про дані елементи оформте у вигляді презентації.

Завдання для самостійного виконання

Додайте до вашого сайта HTML-сторінку з назвою form.html і створіть на ній форму такого вигляду, як наведено на рис. 11.80.

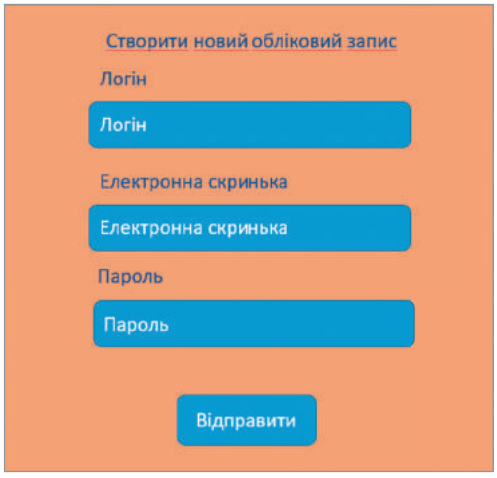


Рис. 11.80. Приклад форми

11.17. Хостинг сайта

Поміркуйте, як інші користувачі можуть отримати доступ до створеного вами сайта.

Сайт створюється насамперед для того, щоб його відвідували користувачі. Для цього якимось чином їм потрібно надати доступ до нього. Звісно, ми можемо розмістити наші веб-сторінки на власному комп'ютері й використовувати його як сервер. Проте це має бути потужний комп'ютер з безперешкодним доступом до нього в режимі «нон-стоп» (тобто 24 години на добу і 7 днів на тиждень), а також виділений канал передавання даних, який надає можливість одночасного звернення до серверу великій кількості осіб. Оскільки описане не завжди можливо, логічно звернутися до організацій, які спеціалізуються на наданні хостингу.

Веб-хостинг — це послуга, що дозволяє приватним особам, підприємствам і організаціям розміщувати в Інтернеті вебсайт або веб-сторінку. Послуги хостингу надають технології та підтримку, необхідні для перегляду веб-сайта або веб-сторінки в Інтернеті.

Веб-сайти розміщуються або зберігаються на спеціальних комп'ютерах, які називають серверами, що обслуговуються або належать службі веб-хостингу, або передані в оренду сторонніми службами, які експлуатують «ферми серверів» або Дата-центри.

Хостинг (англ. *hosting*) — послуга, що включає надання дискового простору, підключення до мережі та інших ресурсів для розміщення фізичної інформації на сервері, що постійно перебуває в мережі (наприклад, Інтернету).

Зазвичай до послуг хостингу вже входить надання місця для поштової кореспонденції, баз даних, DNS-файлового сховища тощо, а також підтримка функціонування відповідних сервісів. Однак вони можуть надаватися і окремо. А власне послуга може бути обмежена розміщенням поштової кореспонденції та відповідного ПЗ (поштовий хостинг), клієнтських файлів (файловий хостинг), виключно відеофайлів (відеохостинг) або інших файлів певного типу та за певних умов.

Провайдерами хостингу можуть виступати як компанії, що спеціалізуються на цих послугах («хостери»), так і великі провайдери інформаційних послуг, що спеціалізуються на інших послугах (такі як Google, Microsoft, Yahoo та ін.).

Розрізняють такі види хостингу:

- віртуальний хостинг;
- віртуальний виділений сервер;
- виділений сервер;
- хмарний хостинг;
- керований WordPress хостинг;
- веб-хостинг.

Поняття хостингу включає широкий спектр послуг із використанням різного апаратного та програмного забезпечення. Зазвичай під цим поняттям, як мінімум, мають на увазі послугу розміщення файлів сайта на сервері, на якому запущене ПЗ, необхідне для обробки запитів до цих файлів (веб-сервер).

Послуги хостингу можуть надаватися у пакеті з іншими інформаційними послугами, такими як реєстрація доменного імені, створення сайта, надання додаткового ПЗ тощо.

2 з 5 користувачів Інтернету відмовляються від повільного завантаження веб-сайта (під повільним розуміється сайт, який завантажується більш ніж 3 с).

1,5 млрд доларів щорічно втрачаються в економіці США через повільне завантаження веб-сайтів, а затримка на 1 с знижує коефіцієнт переходів на 7 %.

Розглянемо види хостингу більш детально (рис. 11.81).

Віртуальний хостинг (*virtualhosting* або *sharedhosting*) — користувачеві надається частина місця на диску для розміщення веб-сайтів. При цьому середовище виконання веб-сервісів єдине для багатьох користувачів, а апаратні і програмні ресурси розподілені між усіма користувачами на одному сервері, де може розміщуватись від 50 до 1000 користувачів.

Перевагами віртуального хостингу є відносно низькі ціни та набір послуг, що є адекватним для функціонування невеликого та оптимізованого сайта. *Недоліками* віртуального хостингу можна вважати те, що через розподілення ресурсів серверу між багатьма користувачами, надмірне споживання цих ресурсів одним сайтом може вплинути на роботу інших.

Віртуальний виділений сервер (VPS або VDS) — послуга, в рамках якої користувачеві надається так званий віртуальний виділений сервер. Спосіб керування операційною системою здебільшого відповідає управлінню фізичним виділеним сервером. Зокрема, права адміністратора, root-доступ, власні IP-адреси, порти, правила фільтрування і таблиці маршрутизації.

Виділений сервер (*dedicated server*) — сервер надається повністю і використовується для реалізації нестандартних завдань (сервісів), розміщення «важких» веб-проектів, які не можуть співіснувати на одному сервері з іншими проектами і вимагають для себе всі ресурси сервера.

Колокація (*collocation*) — надання місця в дата-центрі провайдера для обладнання клієнта (зазвичай шляхом монтажу в стійці) і підключення його до Інтернету. Надає можливість користуватися інфраструктурою дата-центру (системами охолодження повітря, пожежної безпеки тощо).

Колокацією користуються переважно досвідчені клієнти, що мають достатньо навичок для підключення і адміністрування власних серверів, або компанії, що самі надають інформаційні послуги.

Хмарний хостинг (*cloudhosting* або *cloudstorage*) — послуга з розміщення файлів користувача, за якої дані зберігаються на багатьох серверах, що розподілені в мережі (рис. 11.82). Файли зберігаються у так званій «хмарі», що фізично складається із серверів, які можуть знаходитися далеко один від одного, але з точки зору користувача працюють як один потужний віртуальний сервер. Перевагами хмарного хостинга є можливість колективної роботи з даними та відсутність прив'язки до ресурсів одного окремого серверу.

За наведеними даними (рис. 11.83), безперечним лідером є WEB Services Amazon, скорочено AWS (https://aws.amazon.com/), із величезним набором інструментів. Платформа була піонером у цій галузі й завоювала чималий ринок. Можливості Amazon вже сьогодні не мають собі рівних і продовжують зростати в геометричній прогресії.

Microsoft Azure (https://azure.microsoft.com/en-us/) — близький конкурент AWS із надзвичайно розвиненою інфраструктурою. Система існує з 2010 року й розвивається швидкими темпами. Наразі Microsoft Azure являє собою багатогранну

Рис. 11.81. Деякі види хостингу: віртуальний хостинг (*а*); віртуальний виділений сервер (*б*); виділений сервер (*в*)

Віртуальний виділений сервер надає більше можливостей і привілеїв, а часто і більше ресурсів, ніж віртуальний хостинг. Водночас він є і більш дорогим.

Виділений сервер вважається найбільш ефективним і водночас найдорожчим серед усіх видів хостингу. Управління ним потребує від користувача найбільше технічних знань і навичок.

Рис. 11.82. Логотипи платформи хмарного хостингу

складну систему, яка забезпечує підтримку багатьох видів послуг, мовних програм і фреймворків.

Третє місце посідає Google Cloud (https://cloud.google.com/), який з'явився на ринку хмарного хостингу пізніше, проте його провідна роль у галузі штучного інтелекту, машинного навчання й аналітики даних надає значні переваги.

Зараз у світі існує близько 1,94 млрд веб-сайтів. Перша мільярдна позначка була досягнута у вересні 2014 року.

10 найбільших хостингових компаній становлять 24 % ринку веб-хостингу, тобто кожен четвертий веб-сайт користується послугами цих компаній.

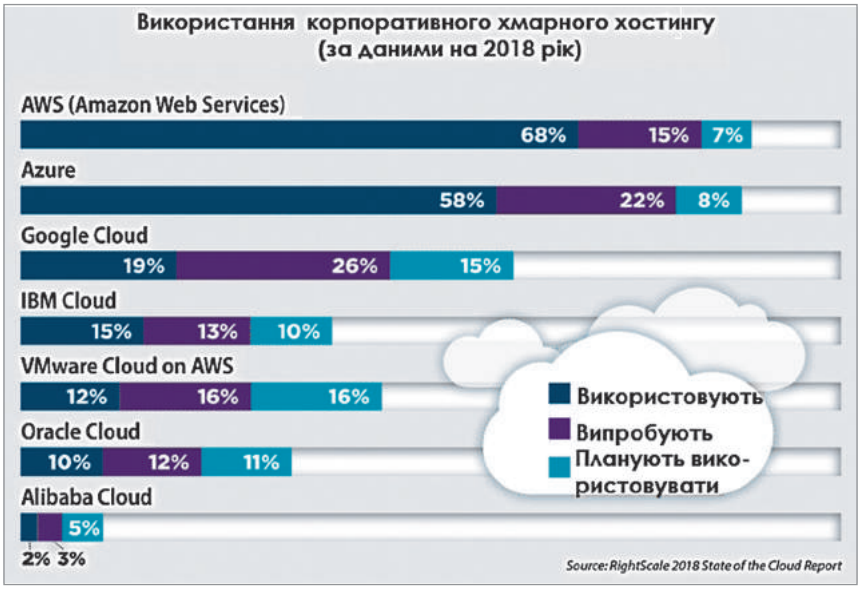


Рис. 11.83. Рейтинг платформ хмарного хостингу

Реселлер хостинг (*resellerhosting*) — хостинг з послугою перепродажу. Користувачеві надається можливість розподіляти дисковий простір і ресурси свого віртуального хостингу або сервера з метою розміщення на ньому сайтів третіх осіб, що можуть бути його клієнтами. Пакет послуг такого хостингу зазвичай включає спеціальне програмне забезпечення для управління клієнтською базою, ресурсами, що надані клієнтам, тощо.

Зазвичай компанія, що надає безкоштовний хостинг, заробляє шляхом показу реклами на сторінках, розміщених на ньому. Такий хостинг, як правило, повільніший від платного, надає тільки базові послуги, ненадійний. Безкоштовний хостинг часто надається для того, щоб через якийсь час перевести користувача на платну основу шляхом погіршення умов користування.

Безкоштовний хостинг використовують приватні особи для своїх домашніх сторінок на початковому етапі їх розвитку. Громадські організації можуть використовувати як платний хостинг, так і безкоштовний. Комерційні організації практично завжди користуються послугами платного хостингу.

Веб-хостинг є одним із типів інтернет-хостингу, який дозволяє окремим особам та організаціям зробити свій сайт доступним через World Wide Web. Сайти компаній, які надають місце на сервері, що належать або орендовані для використання клієнтами, а також інтернет-з'єднання, як правило, розташовані у центрі обробки даних.

Який би тип хостингу не вибрали користувачі, їм необхідна **панель керування** веб-хостингом, яка дозволяє:

- керувати файлами;
- здійснювати статистику трафіку;
- адмініструвати бази даних;
- зберігати облікові записи електронної пошти та конфігурацію;
- керувати протоколом FTP;
- здійснювати переадресацію;
- автоматизувати виконання задач, які повторюються (наприклад, за допомогою популярної утиліти Cron);
- формувати DNS.

Багато інтернет-провайдерів (ISP) пропонують цю послугу безкоштовно для абонентів. Приватні особи та організації можуть також отримати веб-сторінку хостингу від альтернативних постачальників послуг. Особистий веб-сайт хостинг, як правило, безкоштовний або недорогий. Бізнес веб-сайт хостинг часто платний.

Насамкінець, WordPress-хостинг — це різновид загального хостінгу, який спеціально створений для розміщення сайтів на WordPress. При цьому сервер налаштований на найбільш оптимальний режим роботи саме з CMS WordPress. На сайті відразу є заздалегідь встановлені плагіни для кешування, безпеки тощо. Завдяки оптимізованій конфігурації сайт швидше завантажується, також існують додаткові функції, пов'язані з WordPress, такі як додаткові теми WordPress, конструктори сторінок простих перегортань і спеціальних інструментів розробки. Прикладом такого хостингу є перший спеціалізований хостинг в Україні (https://wphost.me/).

Існує ціла низка панелей керування веб-хостингом (**Parallels Plesk Panel**, **DirectAdmin**, **Webmin** тощо). Деякі хостингові компанії використовують один тип панелі керування хостингом, тоді як інші дозволяють обирати один із чотирьох варіантів. Сьогодні більшість панелей керування хостингом також дозволяють одним натисканням кнопки установлювати популярні сценарії, такі як **WordPress**.

Рис. 11.84. Приклад найбільш поширеної панелі керування веб-хостингом **cPanel**

cPanel є найпопулярнішою панеллю керування хостингом онлайн (рис. 11.84) завдяки її легкому налаштуванню. **cPanel** розроблена на початку 1996 року для серверів Linux. Проте компанія пропонує панель управління хостингом під назвою **Enkompass**, створену й для серверів Windows. Крім того, існує додаткова панель керування, яка дає змогу адмініструвати декілька веб-сайтів і налаштувати кожен аспект сервера.

Запитання для перевірки знань

1. Що таке хостинг?
2. Назвіть види хостингів.
3. Опишіть різницю між віртуальним виділеним сервером; виділеним сервером.
4. Пригадайте, що таке SSL.
5. Знайдіть в Інтернеті відомості про українських інтернет-провайдерів. Порівняйте вартість їх послуг. Результати подайте у вигляді інфографіки.
6. Які, на вашу думку, недоліки має безкоштовний хостинг?

11.18. Веб-сервер та бази даних

Що таке протокол HTTP? Опишіть принцип його роботи.

Пригадаємо, що розробка зі сторони браузера називається front-end, а розробка на стороні сервера — back-end. Наразі мова піде про розробку функціоналу на стороні сервера. Серверну сторону веб-сайта найчастіше пишуть такими мовами програмування, як Python, PHP, Java, C#, Ruby, JavaScript.

Щоб завантажити веб-сторінку, браузер відправляє запит до веб-сервера, який приступає до пошуку запитуваного файла у своєму власному просторі пам'яті. Знайшовши файл, сервер його зчитує, опрацьовує як йому потрібно й повертає до браузера. Отже, веб-сервер повинен містити файли веб-сайта, а саме всі HTML-документи і пов'язані з ними ресурси, включаючи зображення, CSS-стилі, JavaScript-файли, шрифти й відео. Веб-сервер забезпечує підтримку HTTP.

Перший у світі веб-сервер, робоча станція **NeXT Computer** з **Ethernet**, 1990. На етикетці написано: «Ця машина є сервером. НЕ ВИМИКАТИ!!»

Веб-сервер — це програма, яка створює і повертає відповіді на запити веб-ресурсів клієнтами.

Порядок дій опрацювання клієнтського запиту:

1) синтаксичний аналіз запиту;
2) перевірка повноважень;
3) зв'язування URL у запиті з ресурсом у файловій системі сервера;
4) побудова відповіді;
5) повернення відповіді клієнту, який звернувся із запитом.

Сервер може генерувати повідомлення-відповідь у різний спосіб. У найпростішому випадку сервер лише витягує файл, асоційований із URL, і повертає вміст клієнту. В інших випадках сервер може викликати сценарій, який зв'язується з іншими серверами або БД для побудови повідомлення-відповіді.

Моніторингові компанії приводять порівняльну статистику, зіставляючи кількість наявних сайтів для різних веб-серверів. І слід зазначити, що у всіх подібних дослідженнях фігурують лише три назви: Apache, IIS (сервер компанії Microsoft) і NGINX (рис. 11.85). Саме на цих трьох китах тримається основна частина мережевого світу. Безумовним лідером понад 20 років залишається Apache.

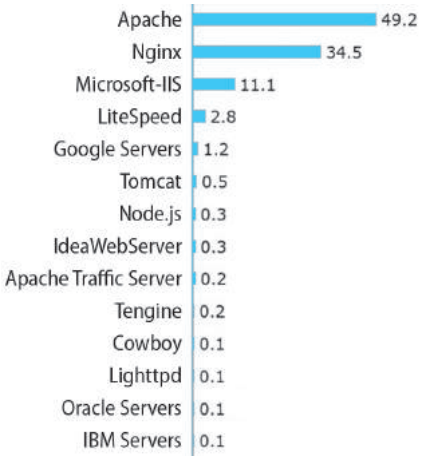


Рис. 11.85. Рейтинг популярності веб-серверів (станом на 2018 рік)

Розглянемо найпопулярніші веб-сервери (рис. 11.86).

Apache — кросплатформний веб-сервер, він добре працює як на Unix-, так і на Windows-серверах. Сервер і клієнт взаємодіють за протоколом HTTP, і Apache відповідальний за безпечне з'єднання між двома машинами. Веб-сервер працює з різними мовами програмування (PHP, Python, Perl та ін.) за допомогою спеціального модуля Apache (mod_php, mod_python, mod_perl та ін.).

Якщо отримано запит із розширенням .php, сторінки обробляються PHP (*Hypertext Preprocessor* або *Personal Home Page*). PHP є мовою сценаріїв, який не залежить від платформи, сценарії вбудовуються в HTML-код. Інтерпретатор PHP виконується під управлінням веб-сервера, інтерпретатор здійснює синтаксичний аналіз і обробку файлів. PHP-файл може містити дані, відправлені користувачем у HTML-формах.

Nginx — веб-сервер, перший реліз якого відбувся в 2004 році. Наразі він набув значної популярності. Nginx було створено для розв'язування так званої «проблеми c10k» — проблеми 10 тисяч з'єднань. Це означає, що веб-сервери, що використовують потоки, не можуть обробляти запити користувачів більше ніж з 10 000 підключень одночасно.

Веб-сервер Nginx має подійно-орієнтовану архітектуру, тобто він обробляє кожен вхідний запит в єдиному потоці. Він використовується сайтами з великою кількістю показів, такими як Netflix, Pinterest і Airbnb.

Apache Tomcat — ще один HTTP-сервер, проте він обробляє додатки Java замість статичних сайтів. Google Web Server, IBM HTTP Server (IHS), Oracle HTTP Server (OHS) — всі ці сервери базуються на Apache і доопрацьовані відповідно компаніями Google, IBM та Oracle.

Сервер Apache Tomcat, опрацьовуючи HTTP-запит, виконує:

1) перетворення URL-запиту в шлях до файла у файловій системі сервера;
2) визначення, чи має запит дозвіл на доступ до файла;
3) ідентифікацію;
4) виклик обробника для створення відповіді;
5) передачу відповіді клієнту;
6) створення запису про запит у журналі

NODE.JS — також кросплатформений веб-сервер. Він працює з одним потоком, і його перевага в тому, що і клієнтська частина, і серверна частина написані однією мовою — JavaScript. Запити до бази даних можна робити, також використовуючи цю мову.

Ознайомимося з цікавими сторінками історії.

Рис. 11.86. Логотипи найпопулярніших веб-серверів

У **PHP** є гнучкі засоби підтримки роботи з БД. Замість того щоб вбудовувати інформацію в HTML-файл, деякі програми можуть генерувати весь ресурс. У цьому випадку URL може розв'язувати різні завдання, такі як доступ до інформації в БД або створення відповіді, зміст якого залежить від клієнта, який звертається із запитом.

У 1994 році співробітник Національного центру додатків для суперкомп'ютерів в Університеті Іллінойсу США (NCSA) Роб Маккул виклав у загальне користування перший веб-сервер — NCSA HTTP daemon. Сервер набув поширення, а влітку 1994 року Маккул залишив університет і розробки припинилися.

Невелика група зацікавлених веб-майстрів почала спільну роботу над продуктом. Спілкуючись у дискусійному листі електронною поштою, вони розробляли нововведення на базі NCSA Server 1.3. Саме вони й заснували Apache Group, яка розробила першу версію Apache-сервера. Сталося це у квітні 1995 року, коли з'явився перший офіційний публічний реліз Apache 0.6.2. Назва «Apache» була пов'язана з широким використанням розробниками технології «програмних латок» (*patches*) і є скороченням від «*a patchy server*».

Робота над сервером не припинялася. Після численних випробувань 1 грудня 1995 року, з'явилася версія 1.0, стійка і надійна.

Протягом усіх цих років і до сьогодні Apache залишається абсолютно безкоштовним. За даними NetCraft, Apache на сьогоднішній момент встановлено на 67 % всіх серверів світу.

Ми вже говорили, що у веб-серверів є досить розвинені засоби підтримки роботи з БД. Пригадаємо, що бази даних (БД) поділяються на реляційні та нереляційні.

До найпопулярніших реляційних БД належать **MySQL**, **PostgreSQL**, **Oracle**.

Щоб комунікувати напряму із реляційною базою, потрібно освоїти ази мови запитів БД — SQL. Проте з часом виникла потреба у складних веб-сайтах, що повинні бути швидкими при великих об'ємах даних, у веб-чатах, можливостей реляційних БД уже бракувало.

Були розроблені так звані нереляційні (NoSQL) БД. Вони діляться на дві підгрупи: перша група розв'язує питання швидкодії під час роботи з великими об'ємами даних, а друга група дозволяє будувати ефективні «живі» веб-застосунки (такі як чати і новини в соцмережах).

До найпопулярніших нереляційних БД належать **MongoDB**, **CouchDB**, **HBase**, **Redis**, **Cassandra**.

Звернемо увагу, що поняття веб-сервера несе два змістових навантаження. З одного боку, це сервер, на якому розміщено веб-сайт з усіма файлами, з яких він складається, а з іншого — так називається програма, що опрацьовує запити клієнтів, які стосуються веб-ресурсів.

Сервер додатків — сервер, на якому створені додатки, які використовують вашу БД, веб-сервіс і т. д. Цей сервер додатків буде розміщувати бізнес-рівень (загорнутий веб-службами), заплановані завдання, служби Windows та ін.

Сервер баз даних — матиме одну або кілька БД, таких як Oracle, SqlServer, MySql та ін.

Програмне забезпечення веб-сервера, додатків і БД може працювати на одному фізичному сервері або розподілятися по кількох фізичних машинах. На більшості великих сайтів є кілька машин; більшість «споживчих» хостингових пакетів запускаються в одному вікні.

Існує багато серверів-додатків. Причому деякі з них орієнтуються на певні категорії веб-сайтів, такі як блоги, вікі-сторінки або інтернет-магазини, деякі, наприклад системи управління контентом, є більш універсальними.

Логічно поділ виглядає таким чином. Веб-сервер обробляє запити HTTP (S) і передає їх «обробникам». У них є вбудовані обробники для запитів файлів — HTML-сторінки, зображення, CSS, JavaScript і т. д.

Якщо необхідно обробити динамічну сторінку, сервер передає її спеціальній програмі, яка і формує остаточну сторінку. Така програма називається сервером додатків. Вона виконує читання коду, що знаходиться на сторінці, формує остаточну сторінку відповідно до прочитаних кодів, а потім вилучає його зі сторінки.

У результаті всіх цих операцій виходить статична сторінка, яка передається веб-сервером, який, у свою чергу, відправляє її клієнтському браузеру. Всі сторінки, які отримує браузер, містять тільки HTML-код.

Сервер додатків надає можливість використовувати БД (приклад).

Програмна інструкція, призначена для отримання даних із БД, називається запитом до бази даних. Запит складається з критеріїв пошуку, записаних за допомогою мови запитів SQL. Текст SQL-запиту розташовується в сценаріях сторінок на стороні сервера або в тегах.

Сервер додатків не може безпосередньо отримати дані з БД, оскільки вони використовують специфічні формати зберігання даних. Для підключення до БД сервер додатків використовує посередника — драйвер бази даних.

Драйвер бази даних являє собою програмний модуль, за допомогою якого встановлюється взаємодія між сервером додатків і БД.

Після того як драйвер установить з'єднання виконується запит до бази даних, у результаті чого формується набір записів, який повертається серверу додатків, і він використовує отримані дані для формування сторінки.

Для різних завдань використовуються різні зв'язки веб-сервера (nginx або Apache) і БД, такі як php_fpm або spawn-fcgi. Проте зазвичай обирається PHP + MySQL.

Розглянемо роботу такої зв'язки на прикладі серверної підтримки розробленої нами форми реєстрації (див. завдання для самостійного виконання § 11.16).

Приклад. Динамічна сторінка може містити програмні інструкції для сервера додатків, виконуючи які, серверу необхідно отримати певні дані з БД і розмістити їх у HTML-код сторінки.

Приклад.
Відбувається надсилання форми з веб-сторінки у вигляді сценарію JavaScript. JS AJAX отримує дані й відправляє їх на HTML-сервер, сервер отримує ці дані, а також певний файл PHP. Файл PHP установлює з'єднання з БД і виконує потрібні дії. Після виконання операції БД повертає результат. PHP відправляє отримані дані назад до AJAX, де залежно від результату остаточно з'ясовуються наступні дії (якщо вже існує такий логін або набраний пароль не відповідає заявленому, то повертається повідомлення про невідповідність і прохання повторити спробу, при підтвердженні облікового запису відкривається сторінка профілю).

Програмне забезпечення БД може працювати на тому самому фізичному комп'ютері, що й веб-сервер. Але зазвичай це перше, що потрібно для розміщення на окремому фізичному обладнанні, коли сайт повинен масштабуватися.

Запитання для перевірки знань

1. Назвіть завдання серверної сторони.
2. Чим відрізняються back-end і front-end?
3. Які мови програмування використовують back-end-програмісти?
4. Наведіть приклади використання серверів.
5. Опишіть послідовність роботи серверів.
6. Пригадайте розвиток реляційних баз даних або знайдіть відомості в Інтернеті.

11.19. Взаємодія «клієнт — сервер»

Пригадайте, що таке клієнт та сервер у мережі. Які функції виконують сервери?

Клієнт і сервер взаємодіють один з одним у мережі Інтернет або в будь-якій іншій комп'ютерній мережі за допомогою різних мережних протоколів, наприклад IP, HTTP, FTP та ін. Протоколів, як ви знаєте, досить багато, і кожен з них дозволяє надавати певну послугу (приклад 1).

Повідомлення, які надсилають клієнти, отримали назву HTTP-запити. Вони містять певні методи, які вказують серверу, як саме обробляти повідомлення. Повідомлення, які надсилає сервер, отримали назву HTTP-відповіді. Крім корисної інформації вони містять спеціальні коди стану, що дозволяють браузеру дізнатись, як сервер зрозумів його запит. Взаємодія клієнтів та серверів знайшла відображення на рис. 11.87.

Веб-сервер може обмежувати доступ користувачів до певних ресурсів. Керування доступом потребує поєднання автентифікації та авторизації.

Автентифікація (перевірка справжності) визначає користувача, який ініціював запит, а під час перевірки повноважень з'ясовується, чи може користувач мати доступ до певного ресурсу. Більшість систем клієнт — сервер автентифікують користувача, запитуючи в нього ім'я і пароль. У цьому випадку на сервері розташовано файл, який містить імена і паролі всіх зареєстрованих користувачів. Причому, з метою захисту інформації паролі можуть зберігатися в зашифрованому вигляді.

Процес автентифікації дає можливість серверу ідентифікувати користувача, який звернувся з HTTP-запитом. Для керування доступом до веб-ресурсів сервер повинен реалізувати стратегію авторизації (перевірки повноважень). Йому необхідно мати ефективний спосіб визначення, які автентифіковані користувачі можуть мати доступ до певного ресурсу.

Подібні стратегії поведінки є складовою частиною налаштування сервера й зазвичай мають вигляд списків керування доступом із переліком користувачів, яким дозволено або заборонено доступ до ресурсу. Як саме створюються списки керування доступом і як зберігаються імена користувачів і паролі, залежить від програмного забезпечення сервера. Ресурси, що вимагають автентифікації, можуть бути розміщені в окремому каталозі з .php. Воно прийнято для сторінок, які обробляються PHP (*Hypertext Preprocessor*).

Замість того щоб вбудовувати інформацію в HTML-файл, деякі програми можуть генерувати весь ресурс. У цьому випадку URL у HTTP-запиті відповідає програмі, а не документу. Програма може реалізувати різні завдання, такі як доступ до інформації в базах даних або створення відповіді, зміст якого залежить від клієнта, що звертається із запитом.

Приклад 1. За допомогою протоколу HTTP браузер надсилає спеціальне HTTP-повідомлення, в якому зазначається, яку інформацію та в якому вигляді він має отримати від сервера. Отримавши його, сервер у відповідь надсилає подібне за структурою повідомлення (або кілька), у якому міститься вся потрібна інформація, зазвичай це HTML-документ.

Рис. 11.87. Взаємодія «клієнт — сервер»

Веб-браузери взаємодіють із веб-серверами за допомогою протоколу передавання гіпертексту (HTTP). Коли ви клацаєте посилання на сторінці, заповнюєте форму або запускаєте пошук, браузер надсилає на сервер HTTP-запит.

HTTP-запит включає:

- шлях, який визначає цільові сервер і ресурс (наприклад, файл, певна точка даних на сервері, що запускається сервіс, та ін.);
- метод, який визначає необхідну дію (наприклад, отримати файл, зберегти або відновити деякі дані тощо):
  - методи передавання даних — GET (отримати певний ресурс, наприклад, HTML-файл, що містить інформацію про товар або список товарів); POST (створити новий ресурс, наприклад, нову статтю на Вікіпедії, додати новий контакт в базу даних);
  - HEAD — отримати метадані про певний ресурс без отримання змісту, як робить GET. Можна використовувати запит HEAD, щоб дізнатися, коли в останній раз ресурс оновлювався, і тільки потім використовувати (більш «витратний») запит GET, щоб завантажити ресурс, який було змінено;
  - PUT — оновити існуючий ресурс (або створити новий, якщо такого не існує);
  - DELETE — вилучити вказаний ресурс.

Раніше вже наголошувалося на відмінностях роботи зі статичним і динамічним сайтами. Розглянемо взаємодію клієнт — сервер на прикладі статичного сайта (рис. 11.88)

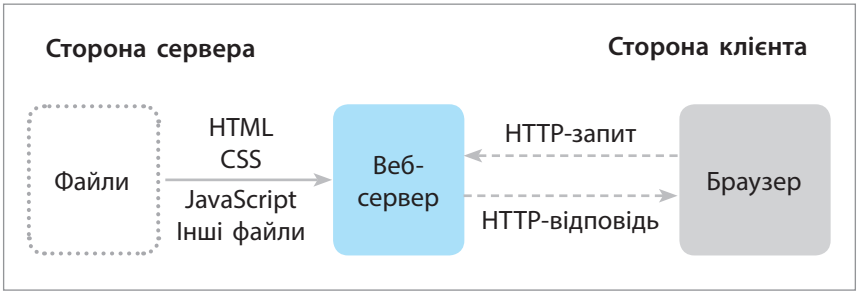


Рис. 11.88. Взаємодія «клієнт — сервер» на прикладі статичного сайта

Коли користувач хоче перейти на сторінку, браузер відправляє HTTP-запит GET із зазначенням URL-адреси його сторінки HTML. Сервер витягає запитуваний документ зі своєї файлової системи і повертає відповідь HTTP, що містить документ, і код стану HTTP «200 OK» (із зазначенням успіху). Сервер може повернути інший код стану, наприклад «404 Not Found», якщо файл відсутній на сервері, або «301 Moved Permanently», якщо файл існує, але був перенаправлений в інше місце.

Серверу для статичного сайта буде потрібно лише обробляти запити GET, тому що сервер не зберігає жодних даних, які модифікуються. Він також не змінює свої відповіді на основі даних HTTP-запиту (наприклад, параметрів URL-адреси або файлів cookie).

Користувач вводить ім'я і пароль на початку сеансу роботи із сервером. Сервер перевіряє ім'я й пароль і зберігає інформацію про користувача на даний сеанс. У певний момент користувач або сервер завершує сеанс. Для подальшого доступу користувачеві необхідно ініціювати новий сеанс роботи із сервером, що вимагає повторного введення імені та пароля.

PHP — мова сценаріїв, яка не залежить від платформи, сценарії вбудовуються в HTML-код. Інтерпретатор **PHP** виконується під керуванням веб-сервера, інтерпретатор здійснює синтаксичний аналіз і обробку файлів. PHP-файл може містити дані, відправлені користувачем у HTML-формах. У **PHP** є гнучкі засоби підтримки роботи з базами даних.

Роботу з динамічним сайтом проілюстровано на рис. 11.89.

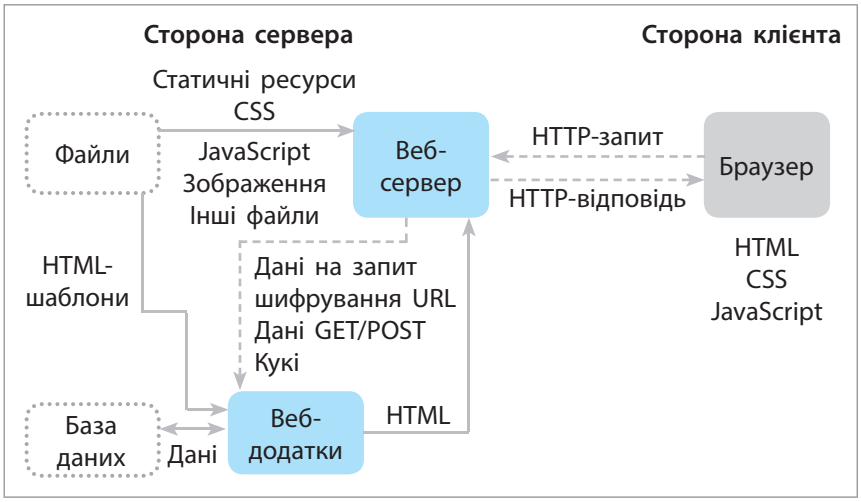


Рис. 11.89. Взаємодія «клієнт — сервер» на прикладі динамічного сайта

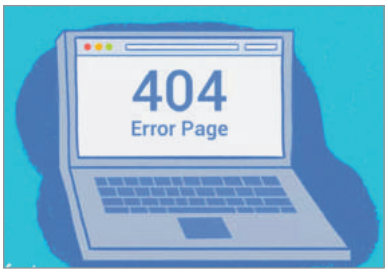


Рис. 11.90. До завдання 5

Розглянемо порядок дій для роботи з динамічним сайтом.

1. Веб-браузер створює HTTP-запит GET на сервер з використанням базової URL-адреси ресурсу. Запит GET використовується, тому що запит — це лише вибірка даних (не зміна даних).
2. Веб-сервер виявляє, що запит є «динамічним» і пересилає його у веб-додаток для обробки (веб-сервер визначає, як обробляти різні URL-адреси на основі правил зіставлення шаблонів, визначених у його конфігурації).
3. Веб-додаток визначає мету запиту, отримує необхідну інформацію з бази даних (використовуючи додаткові «внутрішні» параметри).
4. Веб-додаток динамічно створює HTML-сторінку, поміщаючи дані (з бази даних) у наповнювачі всередині HTML-шаблону.
5. Веб-додаток повертає згенерований HTML у веб-браузер (через веб-сервер) разом з кодом стану HTTP 200 («успіх»). Якщо щось перешкоджає поверненню HTML, він повертає інший код, наприклад «404», щоб вказати, що <такого> посилання не існує.
6. Веб-браузер починає обробляти повернутий HTML, відправивши окремі запити, щоб отримати будь-які інші файли CSS або JavaScript, на які він посилається.
7. Веб-сервер завантажує файли з файлової системи й повертає безпосередньо в браузер.

Операція з оновлення запису в базі даних оброблятиметься аналогічно. За винятком того, що, як будь-яке оновлення, HTTP-запит із браузера повинен бути закодований як запит POST.

Запитання для перевірки знань

1. Що таке автентифікація та авторизація? У чому різниця?
2. Як називають повідомлення, сформовані клієнтом; сервером?
3. У чому різниця між методами **GET** і **POST**?
4. Опишіть механізм автентифікації користувача на сервері.
5. Поясніть повідомлення, наведене на рис. 11.90. Коли воно може з'явитися?
6. Які протоколи (крім HTTP) використовуються під час взаємодії «клієнт — сервер»?

11.20. Валідація сайта та збереження даних форм

Нагадаємо, що різні браузери мають різні алгоритми обробки тегів мови розмітки гіпертексту та селекторів каскадних таблиць стилів, тому завжди існує потреба в перевірці правильності написання коду.

Валідація сайта — перевірка синтаксичних помилок, вкладеності тегів та інші критерії.

Зазвичай валідатори — це сервіси для перевірки сайтів на наявність помилок у структурі документа. Вони перевіряють HTML-код на відповідність певному стандарту, який зазначено на самому початку будь-якої HTML-сторінки першим рядком.

Валідація сайта дозволяє стежити за правильним відображенням сайта в різних браузерах (приклад 1).

Крім того, пошукові системи віддають перевагу сайтам із валідним HTML-кодом, роботи пошукових систем розпізнають html-код за тими самими параметрами, що й браузери (приклад 2).

Якщо ви отримаєте повідомлення про те, що у вашому коді трапляються помилки чи навіть критичні попередження, то потрібно ще раз ретельно оглянути код, перевірити валідацію і провести сканування заново. Такий цикл слід повторювати, доки перевірка не дасть позитивний результат.

Потрібно також звернути увагу на кросбраузерність сайта. В ідеалі сайт повинен однаково відображатися у всіх браузерах. У нашому випадку цю опцію гарантує відповідність нормам W3C.

Більша частина проблем із кросбраузерністю виникає саме через помилки відображення оформлення елементів, заданих за допомогою CSS, оскільки для розв'язання проблеми коректного виведення дизайну сайта в усіх браузерах доводиться застосовувати не зовсім «валідні» (на думку W3C) способи:

- коментарі (коли в коментарях (найчастіше для IE) прописується альтернативне значення властивості, що невидиме для інших браузерів);
- хаки (маються на увазі спеціальні властивості CSS, що дозволяють розв'язати проблему некоректного відображення в одному з браузерів);
- за допомогою JavaScript (зміна стильової властивості елемента через об'єктну модель документа).

Застосування цих способів (особливо хаків) може негативно впливати на валідність усього сайта, разом із тим дозволяє повністю розв'язувати проблему кросбраузерності.

Приклад 1. Якщо не закрити тег або припуститися помилки в коді, у подальшому одна і та сама сторінка може відображатися в різних браузерах по-різному.

Приклад 2. Якщо у вашому коді містяться помилки, наприклад незакриті теги, биті посилання або поламана структура, все це може вплинути на індексацію сайта пошуковими роботами. Як наслідок, грубі помилки в коді можуть знизити позиції ресурсу у видачі. Це означає, що в подальшому валідація сайта зіграє дуже важливу роль у просуванні сайта в SEO і SMM. Більш того, від валідації синтаксису навіть залежить коректність відображення всіх елементів сайта.

Ознайомимося з найбільш поширеними помилками, яких припускаються в коді, на прикладах.

- Тег не закритий — переважна більшість тегів у html є парними, тобто складаються з відкриваючого та закриваючого тегів. Часто в ході верстання чи написання скрипту забувають дописувати закриває тег. А це може призвести до неправильного відображення всього сайта.
- Порушення вкладеності елементів — ця проблема виникає в разі блокового верстання, коли не дотримуються ієрархічної вкладеності всіх блоків . Або інших парних елементів, якщо частина тега відкрита в одному шарі, а закривається в іншому. При цьому може статися порушення структури дизайну сайта. Наприклад:

Неправильно	Правильно
Lorem	Lorem

- Використання атрибута style — у сучасному сайтобудуванні загальноприйнятою нормою є використання для оформлення елементів дизайну каскадних таблиць стилів (CSS). Застарілий атрибут style вживається вкрай рідко: в разі його використання обсяги html коду можуть збільшуватися в рази. Це не лише сповільнює швидкість завантаження всього сайта, а й ускладнює його розуміння і подальше редагування.
- Використання вкладеного CSS — впровадження каскадних таблиць стилів всередину html. Це також збільшує обсяг коду й ускладнює його налагодження. Найкраще стильові описи зберігати в окремих файлах.
- Missing DOCTYPE. Валідатор найчастіше видає це повідомлення, якщо вміст елемента <!DOCTYPE> і сам елемент записані в нижньому регістрі. Необхідно пам'ятати, що даний елемент є залежним від регістру тегом.

Наприклад:

Неправильно	Правильно
<!doctypehtmlpublic"-//w3c//dtdxhtml 1.0 strict//en" "http://www.w3.org/tr/xhtml1/dtd/xhtml1-strict.dtd">	<!DOCTYPE html PUBLIC"-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

- Thereisnosuchelement — такого елемента не існує.

Така помилка виникає, якщо один або декілька тегів записано у верхньому регістрі.

Наприклад:

Неправильно	Правильно
<P><B>lorem</P></B>	Lorem

- requiredattribute "alt" notspecified- обов'язковий атрибут «alt» не вказано.

Як ми вже зазначали, з багатьох причин бажано використовувати атрибут "alt".

Наприклад:

Неправильно	Правильно
<img src="images/girl.png">	<img src="images/girl.png" alt="lorem">

- Missing « « — відсутні лапки.

Така помилка виникає, якщо значення одного або декількох атрибутів записані не в лапках.

Наприклад:

Неправильно	Правильно
<img src=images/girl.png width=189 height=255 alt=lorem>	

Найпоширеніший спосіб перевірити вихідний код сайта на валідацію — це онлайн-сервіси, оскільки вони постійно стежать за нововведеннями та модифікують свої алгоритми.

Посилання на онлайн-сервіси (рис. 11.91) ви можете знайти на головній сторінці офіційного сайта www.w3.org.

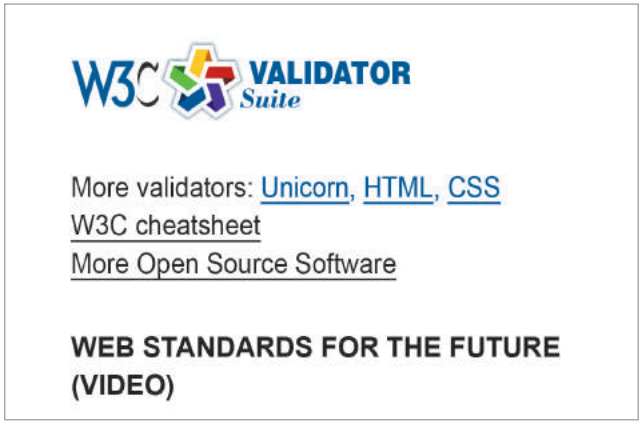


Рис. 11.91. Посилання для перевірки на сайті консорціуму W3C

Валідація (від латин. *validus* — сильний) — визнання чогось законним, дійсним, підтвердження на основі надання об'єктивних доказів того, що вимоги, призначені для конкретного використання або застосування, точно і в повному обсязі зумовлені, а мета досягнута (означення зі стандарту ISO 9000).

Форми також мають правила валідації. Наприклад, можна задати обов'язкові для заповнення поля форми або вказати, що в певні поля потрібно вводити дані тільки певного типу (наприклад, тільки букви або тільки цифри; введення email-адреси; введення url-адреси тощо).

Правила валідації забезпечують правильність заповнення форми відвідувачем сайта. Після того як користувач сайта заповнив форму і відправив дані, вони потрапляють у спеціальний скрипт на сервері — обробник форми. У ньому можуть бути збережені в базі даних або відправлені електронною поштою.

Якщо користувач заповнив веб-форму правильно, без помилок, то після відправки даних на сервер відображається сторінка успішного заповнення форми або відбувається переадресація на заздалегідь задану сторінку.

Якщо ж користувач заповнює форму для опитування або голосування, то після успішного заповнення форми йому відразу ж відображається сторінка із загальними результатами голосування.

Премія Тюрінга (англ. *Turing Award*) — найпрестижніша премія в галузі інформатики, яка щорічно присуджується Асоціацією обчислювальної техніки за видатні досягнення у цій галузі. Премію названо на честь видатного англійського вченого Алана Тюрінга.

Запитання для перевірки знань

1. Що таке валідація сайта?
2. Як валідність коду впливає на індексацію сайта?
3. Наведіть приклади найбільш поширених помилок.
4. Яка організація встановлює правила валідності коду?
5. Наведіть приклади онлайн-сервісів перевірки на валідність.

Завдання для самостійного виконання

1. Перевірте за допомогою будь-якого онлайн-сервісу сторінки, створені на попередніх уроках, на валідність.
2. Проаналізуйте отриманий результат.

11.21. Прикладний програмний інтерфейс

Ви зареєструвались на сайті «Прометеус»? Якщо так, то помітили, що авторизуватись можна за допомогою вашого облікового запису у Facebook та Google?

Це стало можливим завдяки використанню спеціального прикладного програмного інтерфейсу — API.

Прикладний програмний інтерфейс (англ. *Application Programming Interface*, скорочено API) — це сукупність засобів і правил, які вможливлюють взаємодію між окремими складниками програмного забезпечення або між програмним і апаратним забезпеченням.

У галузі веб-розробки поняття прикладного програмного інтерфейсу (API) охоплює низку засобів програмного коду (методи, події та посилання). Розробник може використовувати їх у власних застосунках задля взаємодії з програмним або апаратним оточенням — із додатками веб-переглядача, іншим програмним чи апаратним забезпеченням комп'ютера користувача, або навіть сторонніми сайтами чи службами.

API — це готові модулі коду, які допомагають програмісту в реалізації деяких складних завдань. Зазвичай такі «заготовки» діляться на браузерні і API третіх розробників.

До **браузерних API-інтерфейсів** належать: API-інтерфейс DOM (Document Object Model), модулі геолокації, API Canvas і WebGL, Аудіо та відео API. До **сторонніх інтерфейсів** належать, наприклад, API соціальних мереж Twitter і Facebook.

Ще більше можливостей надає функціонал, доступний як надбудова щодо основних складових JavaScript. Ідеться про API, що істотно розширює набір інструментів розробника.

Коли ми завантажуємо сторінку в браузері, то спочатку обробляються HTML і CSS і лише після цього скрипти.

API надає безліч методів, які можуть використовувати розробники, а також опис того, що вони працюють. Розробнику необов'язково знати, як працює система всередині, він просто може використовувати певний функціонал у своєму додатку.

API-інтерфейси дозволяють розробникам заощадити час, скориставшись вже готовим функціоналом. Це допомагає зменшити кількість виробленого коду, створювати деяку послідовність для різних додатків на тій самій платформі. API-інтерфейси можуть контролювати доступ до апаратних і програмних ресурсів.

Карен Спарк Джон — британська дослідниця, яка розробила технології пошуку, які дозволили користувачам працювати з комп'ютерами, використовуючи слова замість рівнянь і кодів.

Програміст може скористатися API для отримання доступу до функціоналу сторонньої програми. API робить можливим роботу ресурсів, які використовують потенціал і потужність іншого сайта або програми.

Як відомо, на онлайн-сервісах або платформах можна увійти через власні облікові записи в соціальних мережах. Саме це і є використанням API, коли сервіси або додатки використовують бази даних соціальних мереж. Сервіс може отримувати інформацію про користувача і використовувати її у своїх цілях (приклади 1–3).

Використання API скорочує необхідність створювати самостійно складні програми. Замість цього можна

використовувати готові частини існуючих ресурсів, у яких є доступ до потрібної інформації і даних. Щоразу, коли користувач відвідує якусь сторінку в мережі, він взаємодіє з API віддаленого сервера. API — це складова частина сервера, яка отримує запити і відправляє відповіді.

Більшість сучасних сайтів використовують принаймні кілька сторонніх API. Багато задач вже мають готові рішення, пропоновані сторонніми розробниками, будь то бібліотека чи послуга. Найчастіше простіше і надійніше вдатися саме до вже готового рішення.

Як вже було сказано, API — це насамперед інтерфейс, який дозволяє розробникам використовувати готові блоки для побудови програми. У випадку з розробкою мобільних додатків у ролі API може виступати бібліотека для роботи з розумним будинком — всі нюанси реалізовані в бібліотеці, і ви лише звертаєтеся до цього API у своєму коді.

У разі веб-додатків API може передавати інформацію у відмінному від стандартного HTML форматі, завдяки чому їм зручно користуватися під час написання власних програм.

Сторонні загальнодоступні API найчастіше віддають дані в одному з двох форматів: XML або JSON.

XML (англ. *Extensible Markup Language* — розширювана мова розмітки) — запропонований консорціумом World Wide Web Consortium (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками. XML-документ складається із текстових знаків і придатний до читання людиною.

JSON (англ. JavaScript *Object Notation*) — запис об'єктів JavaScript, текстовий формат обміну даними між комп'ютерами. Він базується на тексті й може бути прочитаний людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передавання структурованої інформації через мережу.

Формат JSON знайшов своє головне призначення у написанні веб-програм, а саме при використанні технології AJAX.

JSON досить лаконічний і простий у читанні, дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти на відміну від XML. Сервіси, що надають доступ до даних у XML-форматі, поступово від нього відмовляються.

JSON, що використовується в AJAX, виступає як заміна XML (використовується в AJAX) під час асинхронного передавання структурованої інформації між клієнтом та сервером (приклад 4).

Приклад 1. Amazon пропонує користувачеві книжки, засновані на виборі книжок його друзів у Facebook.

Приклад 2. Сервіс IFFFT дозволяє пов'язувати акаунти користувача в різних онлайн-сервісах і програмах так, що дія в одній програмі викликає дію в іншій програмі.

Приклад 3. Відео, що сподобалися вам у YouTube, можуть автоматично з'являтися на вашому сайті або у ваших соціальних мережах. Це можливо саме завдяки API — коли одна програма використовує дані та інформацію іншої програми. До речі, вбудовування YouTube-відео на свій сайт також можливо завдяки API- сервісу YouTube.

Приклад 4. Візьмемо API Twitter. Інтерфейс цього сервісу може вам видати інформацію про твіти користувача, його читачів і тих, хто його читає, і т. д. Це лише частина можливостей, які будь-хто може втілити, використовуючи API стороннього сервісу або створюючи власний.

На основі API будуються такі речі, як карти 2GIS, всілякі мобільні і десктопні клієнти для Twitter.

Всі їх функції стали можливими саме завдяки тому, що відповідні сервіси мають якісні і детально задокументовані API.

API бувають публічні й приватні.

Публічні API випускаються такими компаніями, як Slack і Shopify, щоб розробники використовували на своїх платформах. Компанії діляться набором вступних параметрів, а розробники використовують, щоб досягти певного результату. Це досить легко, оскільки документація перебуває у вільному доступі.

Приватні API використовуються всередині компанії. Якщо в компанії багато програмних продуктів, то приватне API використовується, щоб програми взаємодіяли між собою. Компоненти API можуть змінюватися за бажанням компанії, тоді як зміни в публічному API може викликати протест.

Фреймворк (*framework* — конструкція, структура) — програмне середовище спеціального призначення, своєрідний каркас, який використовується для того, щоб істотно полегшити процес об'єднання певних компонентів під час створення програм.

Фреймворк — це основа, що дозволяє додавати компоненти залежно від потреб; база, на якій можна сформувати програму будь-якого призначення досить швидко і без особливих труднощів.

Якщо порівнювати динамічну бібліотеку (DLL), яка відрізняється вельми обмеженим функціоналом, і фреймворк, що вважається основою програм — можна виділити суттєву перевагу фреймворків.

Саме фреймворк є сполучною ланкою, яка об'єднує всі використовувані програмні компоненти. Всередині фреймворка часто є необхідні тематичні бібліотеки.

Наведемо характеристики кількох фреймворків.

- Semantic UI (https://semantic-ui.com) — використовується для створення переносимих інтерфейсів. Цей досить молодий фреймворк постійно розвивається. У ньому можна знайти величезну кількість кнопок та інших елементів, необхідних для роботи — зображення, іконки, написи.
- Foundation (https://foundation.zurb.com) — фреймворк, що є одним з досить популярних у сегменті front-end-фреймворків. Останні версії відрізняються поліпшеним функціоналом для сучасних мобільних пристроїв. Завдяки семантичному підходу є можливість використання SCSS, написання більш чистого коду в HTML. Він ідеальний для ситуації, коли потрібно швидке прототипування.

Дуже популярним і затребуваним є фреймворк **Bootstrap** (https://getbootstrap.com). Його презентували ще на початку 2011 року. Його головна перевага — адаптивність (адаптивна верстка). Він дозволяє створювати проекти зі стильним дизайном — проект буде автоматично підлаштовуватися, враховуючи розмір екрана комп'ютера або мобільного пристрою користувача, що переглядає сайт.

До переваг **Bootstrap** належить велика кількість стилів, шаблонів, посторінковий дизайн — це істотно полегшує створення сайта. У **Bootstrap** практично відсутні недоліки. Це не тільки HTML/CSS-фреймворк, у **Bootstrap** також включені плагіни й готові стилі **JS/Jquery**.

Запитання для перевірки знань

1. Наведіть означення API.
2. Запропонуйте власні приклади застосування API.
3. Наведіть переваги використання прикладного програмного забезпечення.
4. Що таке фреймворк? Наведіть приклади.
5. Знайдіть в Інтернеті відомості про використання JSON, зробіть повідомлення.
6. Як можна пов'язати інтерактивність сторінок та використання API?

11.22. Правила ергономічного розміщення відомостей на веб-сторінці

Поміркуйте, чи достатньо створити сайт і наповнити його контентом.

Основна мета створення сайта — його спрямованість на користувача, тому зручність використання, зрозумілість і простота є досить важливими аспектами розробки сайта.

Ергономіка (від грец. *ergon* — робота, *nomos* — закон) — наука, яка вивчає робочі процеси з метою створення оптимальних умов праці, що сприяє підвищенню її продуктивності.

Ергономічний сайт має бути добре пристосований для зручної та безпечної роботи користувача. Щоб цього досягти, потрібно користуватися певними правилами (рис. 11.92).

Юзабіліті (англ. *Usability* — зручність і простота використання) — це підхід, покликаний зробити сайт простим у користуванні, таким, який не потребує додаткового навчання користувача, тобто мати орієнтований на нього інтерфейс.

Розглянемо основні критерії ергономічного розміщення відомостей під час створення сайта:

Критерій	Склад
Лаконічність	Простота викладення
	Неперевантаженість
Чіткість	Ясність
	Структурованість
	Розташування
	Видимість адреси
	Однорідність структури
Швидкість	Час завантаження
	Оптимізовані зображення
Взаємодія	Гіпертекстові посилання
	Сегментація інформації
	Сприяння взаємодії
Адаптивність	Можливість змінити розмір шрифту
Доступність	Доступ до всього
	Взаємодія
	Принцип прозорості
	Підпис
	Вибір кольору
	Правильне використання стилів
	Контраст
	Можливість змінити розмір шрифту

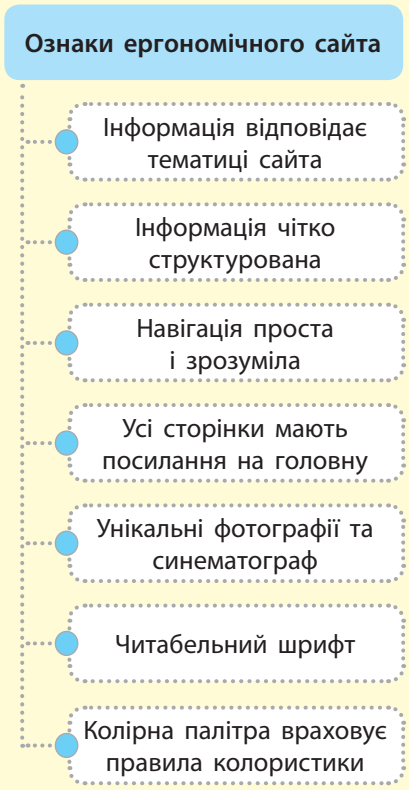


Рис. 11.92. Правила створення ергономічного сайта

Потрібно створити умови, щоб користувач мав змогу інтуїтивно пов'язувати дії, які йому необхідно виконати на веб-сторінці.

Важливо пам'ятати, що вся першочергова інформація повинна бути зліва. Читач знайомиться з нею зліва направо, тому потрібно ретельно все продумати. Таким чином, створення сайта з урахуванням ергономіки може бути визначено як здатність ефективно реагувати на потреби користувачів і забезпечувати їм комфорт під час перегляду сторінки.

Колірна палітра є одним із найважливіших елементів сайта. Існує безліч онлайн-сервісів, які допомагають дібрати палітру кольорів сайта, що відповідає правилам колористики (рис. 11.93).

Деякі загальні цілі юзабіліті:
- подавати інформацію в зрозумілій і стислій формі;
- створювати користувачам можливість робити вибір найочевиднішим шляхом;
- усувати будь-яку двозначність щодо наслідків дій (наприклад, кнопка видалити/покупка);
- розміщувати важливі елементи у відповідній ділянці на веб-сторінці або веб-додатку.

Рис. 11.93. Логотипи деяких онлайн-сервісів

Найпопулярніші онлайн-сервіси можна знайти за посиланнями:
- ColorExplorer (http://colorexplorer.com/)
- cOLOROTATE (http://mobile.colorotate.org/)
- ColorBlender (http://www.colorblender.com/)
- ColorWizard (http://www.colorsontheweb.com/Color-Tools/Color-Wizard)

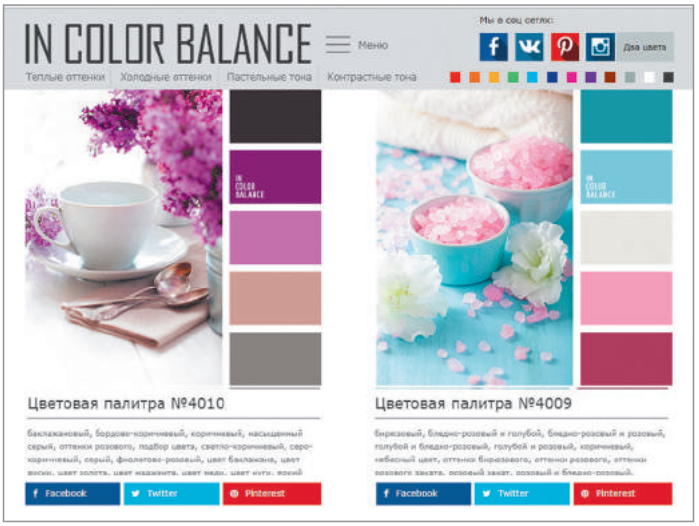


Рис. 11.94. Палітри **In color balance**

Запитання для перевірки знань

1. Що таке юзабіліті; ергономіка? Що, на вашу думку, їх відрізняє?
2. Наведіть правила створення ергономічного сайта.
3. Які онлайн-сервіси добору палітри ви знаєте?
4. Чому важливо правильно добирати кольори?
5. Виберіть сайт і проаналізуйте його з точки зору юзабіліті.
6. Проаналізуйте тренди поточного року та оцініть їх з точки зору ергономіки.

11.23. Пошукова оптимізація та просування веб-сайтів

Яким чином, на вашу думку, на першій сторінці пошуку з'являються ті чи інші сайти?

Будь-який сайт створюється для того, щоб його відвідували якомога більше користувачів.

SEO (від англ. *Search Engine Optimization* — пошукова оптимізація) — маркетингове поняття, що охоплює цілий комплекс заходів. Це процес коригування HTML-коду, структури та текстового наповнення (контенту) сайта; контроль зовнішніх чинників на відповідність вимогам алгоритму пошукових систем.

Мета SEO — підвищення рейтингу сайта в пошукових системах (Google, Bing, Yahoo) та залучення трафіку.

Завдання SEO — підняти веб-ресурс у топ за результатами пошукової видачі за конкурентними запитами користувачів.

Складові SEO наведено на рис. 11.95. Що вищою є позиція сайта в результатах пошуку, то більшою є ймовірність переходу відвідувача на нього з пошукових систем, бо зазвичай люди ідуть за першими посиланнями.

Пошукова оптимізація може бути постійним джерелом збільшення кількості відвідувачів, адже 90 % користувачів знаходять нові сайти через пошукові системи. 55 % онлайн-покупок здійснюються на сайтах, які знайдено через пошукові системи.

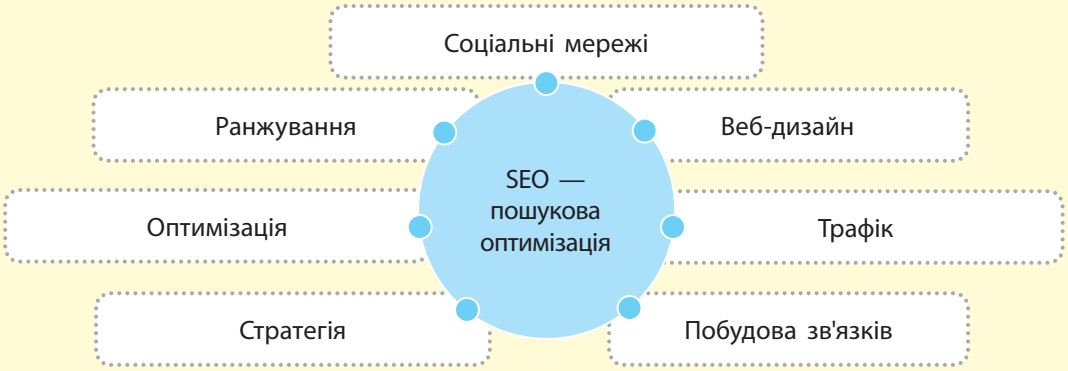


Рис. 11.95. Складові SEO

Роботи з позиціонування сайта в пошукових системах — це один із найважливіших заходів щодо залучення цільової аудиторії. Особа, яка проводить роботу з оптимізації веб-сайтів, називається *оптимізатором* (SEO-Manager).

Основною пошуковою системою у світі нині є Google. Алгоритм розрахунку авторитетності, який використовує Google, — це PageRank. Він застосовується до бази документів, пов'язаних гіперпосиланнями, — призначає кожному документу чисельне значення, що характеризує його «авторитетність» (що більше посилань на сторінку, то важливішою вона є).

Популярність **Google** у більшості країн становить понад 60 %, в окремих європейських країнах — понад 90 %.

Сервіси для пошуку інформації від українських розробників:

- Sova.com.ua
- Gala.Net
- Online.ua
- MetaPing
- Аванпорт
- Мета
- UAport
- Атлас UA
- I.ua
- Ukr.net
- Meta.ua
- Poshukach.com
- Shukalka.com.ua

За введеними словами пошукова система шукає сторінки, зіставляє їх і ранжує, визначає релевантність конкретної сторінки за введеним запитом (тобто наскільки повно документ відповідає критеріям, зазначеним у запиті користувача).

Марісса Енн Маєр відома тим, що у 1999 році стала першою жінкою-інженером Google, тривалий час посідала керівні посади в компанії. У 2012–2018 роках була президенткою і головною виконавчою директоркою компанії Yahoo!, зараз працює над новим проектом.

Пошукова система — повністю автоматизований механізм, який глибоко сканує всі задані сервери (відкриті для сканування) і збирає індекс-інформацію про те, що і де (на якій веб-сторінці) виявлено.

Зібрана інформація вноситься до бази даних пошукової системи, де алгоритм із ранжування реалізується у два етапи. Спочатку по сайтах «проходить» так званий «швидкобот» та індексує їх для того, щоб додавати новини на видачу пошукових систем, а потім (здебільшого протягом доби) — основний бот, який уже повністю індексує статтю.

Пошуковий робот (*webcrawler, bot, webrobots, webspider* — бот, павук) — це спеціальна програма, що є складовою частиною пошукової системи та призначена для перебирання сторінок Інтернету з метою занесення інформації про них у базу даних пошукової системи.

Оптимізація та просування сайтів у пошукових системах (рис. 11.96) мають забезпечити використання ключових слів у контексті, заголовках і мета-даних.

Рис. 11.96. Робота пошукової системи

Пошукова оптимізація передбачає такі етапи.

Етап 1	Збирання семантичного ядра. Під час добирання всіх слів вибираються ключові словоформи, що додаватимуть відвідувачів на сайт. Водночас здійснюється аналіз ринку сайтів, які наявні в топ-видачі. Збирання ключових слів для написання внутрішніх текстів можна виконати за допомогою як платних, так і безкоштовних програм.
Етап 2	Написання на основі семантичного ядра контенту, що відповідатиме запитам користувачів і розв'язуватиме проблеми відвідувачів сайта. Водночас кількість ключових слів у співвідношенні до загальної кількості тексту не має перевищувати 3–5 %.

Просування сайта — комплекс заходів щодо збільшення відвідуваності веб-ресурсу цільовими відвідувачами

Метою будь-якого просування є збільшення співвідношення відвідувачів сайта, які вчинили очікувану дію, до всіх відвідувачів (подається у відсотковому вираженні).

Власник веб-ресурсу (комерційна компанія, державна організація, соціальна мережа, ігровий майданчик, клуб за інтересами тощо) намагається, щоб його сайт було «видно» користувачам Інтернету, щоб була змога зібрати якомога більше відвідувачів. Таким чином він прагне забезпечити собі потрапляння на перші сторінки (а ліпше — на перші рядки, у топ) пошукової видачі. Розв'язання цього завдання забезпечує процес SEO-просування.

Існує низка прийомів, які дозволяють маніпулювати пошуковою системою. Вони розрізняються за своєю коректністю й легальністю. У зв'язку з цим виникло три класи пошукової оптимізації сайтів: чорна, сіра та біла. Ознайомимося з їхнім призначенням.

Який відсоток відвідувачів інтернет-магазину від їх загальної кількості здійснили покупку? Аналіз дій користувачів на сайті здійснюється за допомогою спеціального програмного забезпечення. Для **Google** — це **Google Analytics**. За даними досліджень, на другу сторінку переходять не більше ніж 85 % користувачів, далі другої — не більше від 10 %.

Оптимізація	Призначення
Чорна	Набір прийомів, які характеризуються явною некоректністю. Багато з них заборонені, деякі створені для введення пошукової системи в оману
Сіра	Не заборонені, але потенційно некоректні прийоми. Якщо чорну або сіру оптимізацію буде виявлено пошуковою системою, до сайта будуть застосовані штрафні санкції чи навіть бан
Біла	Використовує легальні професійні методи, що доповнюють один одного і дають стабільний результат. Спрямована на те, щоб і відвідувачі, і пошукові машини ставили сайту високу оцінку

Існує кілька розповсюджених методів просування сайта.

До наймолодших і перспективних методів з використанням можливостей, що відкриваються соціальним мережам, належать SMO (*Social Media Optimization*), або SMM (*Social Media Marketing*).

Методи SMO (SMM) просування сайта засновані на самостійній передачі даних від одного користувача до іншого за допомогою сервісів соціальних мереж (рис. 11.97).

Рис. 11.97. Способи SMO (SMM) просування сайта

Яскравим прикладом вірусного маркетингу є розміщення цікавих відеороликів на **YouTube** і подібних до нього сервісах, де як джерело відео вказується ресурс, що просувається.

Ще одним методом просування сайта є так званий вірусний маркетинг — тут головними розповсюджувачами інформації є самі користувачі. Один користувач передає інформацію з посиланням на ресурс знайомим, кожен із яких — кільком своїм. Таким чином спрацьовує ефект «снігової лавини», що приносить результат у вигляді тисяч, а часом і сотень тисяч унікальних відвідувачів.

Основним інструментом вірусного маркетингу є створення цікавого, несподіваного, креативного контенту, який гарантовано зможе зацікавити максимальну кількість користувачів таким чином, аби змусити їх поділитися інформацією з іншими.

Запитання для перевірки знань

1. Що таке SEO?
2. Назвіть завдання SEO.
3. Що таке пошуковий робот? Наведіть схему роботи робота Google.
4. Назвіть класи оптимізацій сайтів.
5. Що таке SMM?
6. Які ви знаєте способи просування сайта? Опишить їх.

Тест 11

Тестове завдання з автоматичною перевіркою результату на сайті interactive.ranok.com.ua

Розділ 4. ПАРАДИГМИ ТА ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

12.1. Уніфікований процес розробки програмного забезпечення

Пригадайте складові інформаційної системи та їхнє призначення. Назвіть види програмного забезпечення персонального комп'ютера. Наведіть приклади програмних засобів, які ви використовували на уроках інформатики.

Розроблення програмного забезпечення (ПЗ) — це складний процес, у якому беруть участь багато фахівців і який часто орієнтований на віддалену роботу.

Розробка ПЗ як результат роботи колективу починається з аналізу вимог до нього. Під час планування проектної роботи з розробки ПЗ застосовують системний підхід (системний аналіз). Він полягає в здійсненні аналізу та уточненні завдання, розробці загальних стратегій роботи, визначенні етапів виконання проекту.

Для злагодженої роботи та швидкого реагування на різні ситуації в ІТ-фірмах формують систему діяльності, що сприятиме ефективній організації розробки ПЗ.

Системний аналіз розробки програмного забезпечення — це дослідження вимог до ПЗ, його можливостей і представлення в загальному вигляді з подальшою деталізацією.

У результаті аналізу створюється ієрархічна структура, кожний наступний (нижчий) рівень якої описує певний аспект ПЗ більш детально, ніж попередній (вищий).

Під час аналізування розглядаються й коригуються вимоги замовника, визначається, як ці вимоги будуть реалізовані, формулюються підзадачі та способи їх розв'язування. Планування та реалізація проекту є процесом, спланованим за певною методологією.

Методологія — це принципи, сукупність ідей, методів і засобів, які визначають підхід до розробки програмного забезпечення.

Існує кілька методологій створення ПЗ. Кожна з них визначає своє планування створення ПЗ, яке передбачає певні етапи. На рис. 12.1 наведено етапи планування розробки ПЗ у загальному вигляді.

Залежно від обраної методології взаємодія між етапами планування й виконання різних етапів відбувається по-різному.

Рис. 12.1 Загальне подання проектування роботи з розробки ПЗ

Етапи розробки ПЗ ще називають *стадіями розробки*. У всіх методологіях їх зміст загалом однаковий. Ознайомимося з ними докладніше.

Етап 1	З **етапу аналізування вимог** до розроблюваного ПЗ, їх систематизації, документування, а також виявлення й розв'язування суперечностей починається будь-який проект з розробки ПЗ. Створюється певна документація, що описує відповідність створеного ПЗ вимогам, на яких етапах будуть реалізовані вимоги, правила встановлення й використання ПЗ. Проте за деякими методологіями в документації можливі зміни в процесі життєвого циклу ПЗ (час від моменту створення ПЗ до вилучення з експлуатації)
Етап 2	На **етапі проектування** (його називають також *стадією дизайну й архітектури*) програмісти й системні архітектори, керуючись вимогами, розробляють високорівневий дизайн системи. Схематично занотовуються характеристики ПЗ
Етап 3	На **етапі написання коду** на основі попередньо визначених вимог програмісти-розробники створюють програми; системні адміністратори налагоджують взаємодію між програмами, front-end-програмісти розробляють користувацький інтерфейс і можливості взаємодії ПЗ із серверами
Етап 4	На **етапі тестування** здійснюється перевірка правильності роботи кодів кожної складової ПЗ, роботи ПЗ на відповідність вимогам у цілому. Тестування відбувається на основі тестів, написаних програмістами-тестувальниками
Етап 5	На **етапі впровадження й підтримки ПЗ** у нього вносяться зміни для оновлення в процесі використання

Як ми вже знаємо, існує низка методологій створення ПЗ. Ознайомимося з найпоширенішими.

Розглянемо сутність каскадної та ітераційної методологій.

Каскадна методологія (англ. *Waterfall model*), її ще називають «крок за кроком», «водоспад», описує послідовний процес виконання всіх етапів проекту (див. рис. 12.1).

До кожного наступного етапу переходять тільки після повного завершення попереднього — зворотний напрямок неможливий (саме тому метод отримав назву «водоспад» — за аналогією потоку води в одному напрямку). Внесення змін до виконаного етапу можливе тільки повторенням усіх етапів починаючи спочатку.

Каскадною методологією користуються у випадку, коли основні вимоги остаточно визначені, зрозумілі й зафіксовані. Зазвичай нею користуються для виконання невеликих проектів.

Ітераційна методологія (від слова «ітерація» — повторення) є більш гнучкою: розробка ПЗ базується на кількох циклах повторення всіх етапів (рис. 12.2 у прикладі).

На першому етапі проекту за ітераційною методологією допускається неповний перелік вимог, адже визначення всіх вимог часто є задачею, яку неможливо розв'язати. На кожному циклі кожен етап виконується паралельно з аналізом отриманого результату. Це дозволяє вдосконалювати подальші етапи та корегувати вимоги при кожному повторі виконання проекту.

За такої організації створюються більш сприятливі умови для замовника, бо після кожної ітерації замовник може спостерігати результат і розуміти, задовольняє він його чи ні, вносити зміни до вимог.

Приклад. Ітераційну методологію використовують у випадках, коли вимоги до ПЗ зрозумілі, хоча деякі деталі можуть доопрацьовуватися в ході розробки (рис. 12.2), а також для реалізації дуже великих проектів. Показовим прикладом є система розпізнавання голосу. Її головне завдання — ідеальне розпізнавання — ще не досягнуто, хоча результат кожної наступної ітерації покращується.

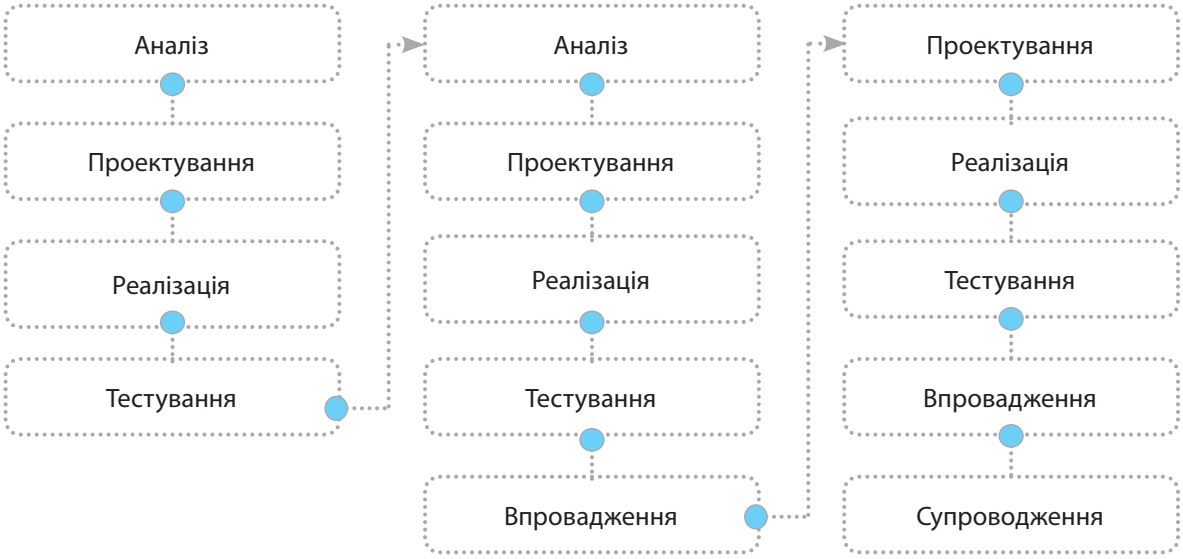


Рис. 12.2. Приклад структури процесу розроблення ПЗ

Вибір певної методології залежить від специфіки і складності проекту та інших чинників, на які впливає кількість розробників у групі, їхні особисті якості й кваліфікація або стабільність процесів у компанії та ін.

У випадку каскадної методології етап аналізу вимог закріплюється в документі — специфікації вимог до ПЗ (Software Requirement Specification, SRS), оформлення якого має бути завершено до переходу на наступний етап.

Запитання для перевірки знань

1. Як ви розумієте поняття методології?
2. Назвіть загальні етапи проектування розробки ПЗ.
3. Поясніть особливості роботи програмістів-розробників і програмістів-тестувальників.
4. На основі чого відбувається тестування ПЗ?
5. Назвіть відомі вам моделі методологій і поясніть, чим вони відрізняються.
6. Наведіть приклади використання різних методологій розробки ПЗ.

Існують й інші методології розробки ПЗ, наприклад **V-модель** (від англ. validation and verification). Вона передбачає тестування всіх параметрів ПЗ на відповідність і запланованим вимогам, і тим, якими вони мають бути в дійсності. Наприклад, інтегроване ПЗ для механізмів управління аварійними подушками безпеки в транспортних засобах, мобільний додаток для європейського стільникового оператора, який економить витрати на роумінг під час подорожей.

12.2. Програмні інструменти для проектної роботи

Чи можна в проекті з розробки програмного забезпечення використовувати роботу з віддаленим доступом і залученням фахівців у територіально розподілених командах?

Ефективність роботи з розробки та експлуатації програмного забезпечення (ПЗ) залежить від багатьох чинників: планування загальної роботи колективу, поінформованості учасників проекту про стан його виконання, внесення змін. У ході використання ПЗ відбувається удосконалення й упровадження нових версій.

Для автоматизації процесу планування й аналізу відповідності реалізації проектної роботи запланованій роботі використовують спеціальні програмні засоби, які називають системами комунікацій, або програмними інструментами.

Для кожного типу роботи колективу існують різні програмні системи комунікацій. Розглянемо особливості найбільш поширених із них (на рис. 12.3 наведено їх логотипи).

Asana і Trello — системи, що використовують для відстеження стану виконання поточних завдань проекту. Asana розрахована на введення таких даних по окремих завданнях, як назва, опис завдання, дата виконання й відповідальна особа. Спілкування розраховане за допомогою електронної пошти. Trello подібна до дошки, на якій прикріплені стікери із завданнями, вона використовує такі характеристики завдань: Заплановано, В роботі, Виконано.

Smartsheet і Мегаплан — системи, придатні для базового управління проектом і організації діяльності невеликого підприємства. Вони базуються на табличній формі подання інформації, подібної до Excel: часто для відстеження виконання завдань проекту користуються графіком, побудованим як діаграма Ганта (ви побудуєте її в Excel під час виконання практичної роботи).

Smartsheet і Мегаплан є онлайн-інструментами, які пропонують необмежену кількість таблиць, збереження файлів, журнал змін, автоматизацію робочих процесів у вигляді діаграм Ганта, а також мобільну версію.

JIRA і Clarizen — системи, що застосовують для всіх типів завдань, починаючи від організації роботи малої групи до великого підприємства. JIRA швидше застосовна для організації підтримки і розробки, Clarizen — для управління портфелем проектів. Крім того, існують також такі системи, як Redmine, Адванта, Comindwork, Liquidplanner, Innotas, AtTask та ін. (див. рис. 12.3).

Одним із ключових моментів організації процесу розроблення ПЗ є комунікаційний процес між учасниками проектної роботи, який набуває особливої важливості, якщо в проекті беруть участь фахівці у віддаленому доступі.

Завданнями процесу комунікацій є формування й підтримка постійного оперативного зв'язку між членами команди,

Рис.12.3. Логотипи найбільш поширених систем комунікацій

визначення, яка інформація передаватиметься кожному з учасників проекту, проведення нарад з планування контролю та роботи, організація онлайн-навчання та посилення зв'язків між членами команди.

У територіально розподіленій проектній групі фахівцю важливо знати, як і коли він може звернутися до інших членів групи, керівника для розв'язання проблем, які в нього можуть виникнути.

Залежно від завдання розрізняють кілька типів комунікаційних процесів (рис. 12.4).

- За ознакою віддаленості: *внутрішні* (комунікації між членами учасників проекту); *зовнішні* (забезпечення зв'язку з керівництвом ІТ-компанії, замовником, зовнішніми організаціями тощо).
- За координацією спілкування: *формальні* (визначені в плані комунікацій проекту, наприклад документування роботи над проектом); *неформальні* комунікації (особисті контакти між членами команди).
- За організаційною спрямованістю: *горизонтальні* (між учасниками проекту одного статусу); *вертикальні* (з урахуванням статусу учасника проекту).

У свою чергу вертикальні комунікації діляться на *низхідні* (від керівника проекту через керівника команди до безпосередніх розробників більше стосуються завдань проекту) і *висхідні* (зворотні комунікації порівняно з низхідними в основному містять інформацію про хід виконання завдань проекту).

- За формою подання комунікації: *аудіо-*, *відео-*, *текстові повідомлення*.

До інструментів комунікацій належать планувальники завдань (Task-менеджери), програми для спостереження за часом виконання роботи (тайм-трекери), системи моніторингу для забезпечення безпечної роботи людей та працездатності об'єктів, сервіси онлайн-листування (e-mail), відеоконференції, відеозв'язки, групові чати, веб-конференції (текст, аудіо, відео), групи в соціальних мережах, електронні дошки оголошень та ін.

Рис. 12.4. Класифікація комунікаційних процесів

Приклад. Порівняємо процес створення ПЗ з розв'язанням задачі на заняттях з програмування: написаний код програми можна вдосконалювати кілька разів, і інколи виникає потреба повернутися до попередніх версій коду. Так відбувається і з новим ПЗ. Прикладом є будь-яка операційна система, кожна версія якої більш досконала і зручна для користувачів, або програми офісного додатку.

У процесі створення ПЗ необхідно зберігати попередні версії програми. Щоб не задіяти безліч файлів, користуються спеціальним інструментом — системою контролю версій (СКВ). Вона містить протокол змін у файлі або в кількох файлах для надання можливостей працювати з попередніми версіями та викликати їх надалі.

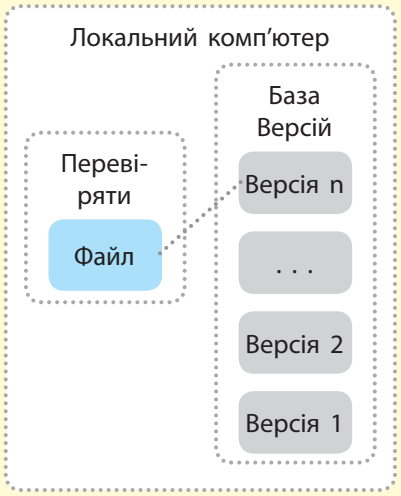


Рис. 12.5. Схематичне зображення локальної СКВ

Існують різні типи СКВ: локальні, централізовані, розподілені (на рис. 12.5–12.7 подано їхні схематичні зображення).

Локальною СКВ є база даних для збереження всіх змін файлів, до яких має доступ СКВ. Прикладом є система rcs, робота її базується на зберіганні наборів патчів (різниць між файлами) у спеціальному форматі на диску та на можливості відтворення вигляду будь-якого файла.

Централізовані СКВ створені для опрацювання змін у файлах, розташованих на різних персональних комп'ютерах. Зазвичай їх робота основана на збереженні всіх версії файлів на сервері, до якого звертаються клієнти, щоб отримати необхідні копії. Прикладами є системи CVS, Subversion і Perforce.

Розподілені СКВ відрізняються від централізованих тим, що клієнт, окрім копії необхідних версій файлів, отримує із сервера всі версії збережених файлів (створюється дзеркало репозеторія). Саме це є перевагою розподілених СКВ.

Якщо сервер виходить із ладу, у централізованих СКВ губляться всі дані щодо змін у файлах, а в розподілених СКВ версії файлів відновлюють із репозиторія клієнтів.

Розподілені СКВ можуть надавати доступ до кількох віддалених репозиторіїв, що уможливлює одночасну співпрацю розробників. Прикладами є системи Git, Mercurial, Bazaar або Darcs.

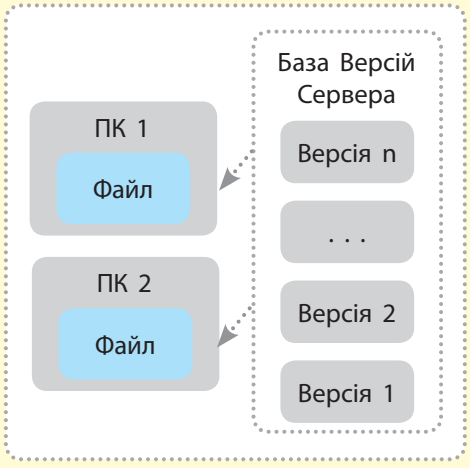


Рис 12.6. Схематичне зображення централізованої СКВ

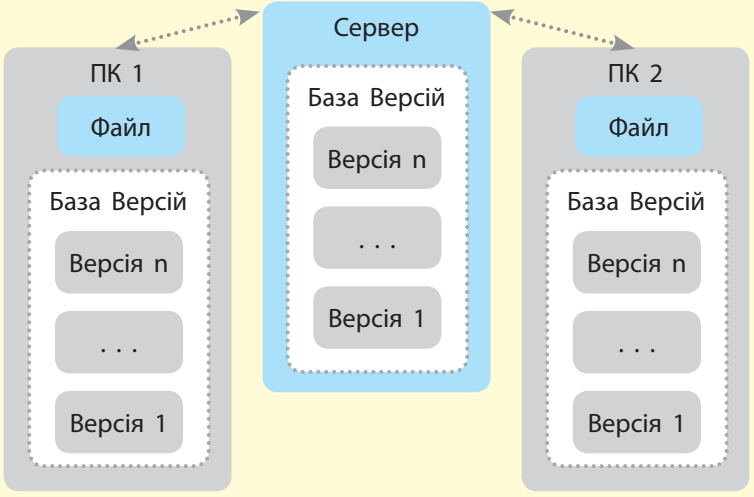


Рис. 12.7. Схематичне зображення розподіленої СКВ

Запитання для перевірки знань

1. Які є інструменти планування проекту?
2. Перелічіть задачі проекту, які розв'язуються у процесі комунікацій.
3. Назвіть типи комунікаційних процесів.
4. Дайте означення системи контролю версій.
5. Які ви знаєте типи СКВ?
6. У яких випадках доцільно використовувати певний тип СКВ?

12.3. Візуальне моделювання архітектури програмного забезпечення

Наведіть означення понять моделювання, моделі. Пригадайте принципи об'єктно-орієнтованого програмування: дайте визначення понять класу, інкапсуляції, поліморфізму, наслідування.

У створенні програмного забезпечення (ПЗ) і в зручному супроводі в процесі його експлуатації найважливішу роль відіграє побудова правильної структури.

Як відомо, створене ПЗ може оновлюватися, а в розробці беруть участь багато працівників (і кожна група виконує певну задачу). З огляду на це, планування ПЗ доцільно починати з його опису в загальному вигляді як системи, що складається з окремих компонентів.

Архітектурою програмного забезпечення є його структура, що складається з програмних компонентів, їхніх інтерфейсів і опису засобів об'єднання компонентів в єдину систему.

Архітектурний стиль визначає засоби організації компонентів та їх взаємозалежностей у системі ПЗ. Архітектура завжди описується в документації ПЗ, що в подальшому дозволяє використовувати архітектурний стиль певного проекту в інших проектах як шаблон.

Архітектурою визначається *набір правил щодо організації п'яти структурних компонентів* в єдину систему: сукупність структурних компонентів системи і зв'язків між ними; взаємозв'язок окремих елементів системи; ієрархію підсистем, об'єднаних структурними компонентами; архітектурний стиль (використовуються методи і засоби опису архітектури, а також архітектурні образи).

Для одного ПЗ існують кілька видів подання його архітектури залежно від мети використання. Програмне забезпечення є складною системою, яка у свою чергу є сукупністю взаємопов'язаних підсистем, об'єднаних одним завданням до функціонування. Для розуміння складових системи, їх функціональних особливостей створюють кілька моделей архітектури, у яких відображені властивості, суттєві для поставленого завдання, та їхня взаємодія.

Як у моделюванні об'єкта можна отримати кілька моделей залежно від мети дослідження (пригадайте моделі кровотворної системи людського організму, модель — скелет, карта м'язів людини), так і для одного ПЗ існують кілька видів подання його архітектури залежно від мети використання.

Архітектура ПЗ може мати різні форми подання, найбільш наглядною з яких є візуальна.

Візуальним моделюванням називають метод розробки ПЗ, який для візуалізації, опису, проектування, документування архітектури ПЗ використовує графічні моделі.

Вперше поняття архітектури зустрічається в 1968 році в описах науково-дослідних робіт Едсгера Дейкстри та на початку 1970-х років у Девіда Парнаса.

Едсгер Вібе Дейкстра (на фото зліва) — нідерландський науковець у галузі комп'ютерних наук. У 1972 році був відзначений премією Тюрінга за вагомий внесок у розвиток мов програмування.

Девід Л. Парнас — канадський учений у галузі інженерії програмного забезпечення.

Може бути подання функціональних можливостей ПЗ, опис структури програмних компонентів у середовищі реалізації, опис фізичного розміщення програмних компонентів, відображення логічної організації компонентів у систему тощо.

Візуальне моделювання застосовується в процесі проектування та аналізування системи перед написанням коду, а також документування, тестування, розроблення вимог до ПЗ. Як приклад можна навести блок-схему програми розв'язування певної задачі. У блок-схемах використовуються блоки, кожний із яких має своє призначення і логічний зміст. А реалізувати блочне розв'язання задачі можна в будь-якому середовищі програмування.

Макет сайта, для зображення якого скористалися графічним редактором, є візуальною моделлю сайта. Візуальне моделювання здійснюється за допомогою інструментальних засобів, до яких належать і мови візуального моделювання.

Мова візуального моделювання (або **візуальна мова**) — це сукупність формалізованих наборів графічних символів і правил побудови з них візуальних моделей.

Універсальна мова моделювання UML є засобом моделювання (не програмування) для створення моделей будь-якого рівня абстракції для ПЗ будь-якого призначення.

Основою UML-моделі є спеціальні графічні конструкції, які називаються *діаграмами*. Кожний тип діаграми має своє призначення і дає уявлення про архітектуру ПЗ з різних точок зору: діаграми, які показують статичну структуру програми, діяльнісні аспекти системи, фізичні аспекти функціонування системи (діаграми реалізації). Діаграми містять елементи, що взаємодіють між собою, — побудова діаграм заснована на моделі «сутність — зв'язок».

Мова UML містить *нотації* — правила побудови діаграм (оскільки мова не належить до мов програмування, то поняття нотації аналогічне поняттю синтаксису при написанні програм). Елементами нотації є графічні зображення: фігури, лінії, написи.

При проектуванні складного ПЗ використовують технологію CASE (англ. *Computer Aided Software Engineering* — комплекс інструментів і методів програмної інженерії для проектування ПЗ). CASE забезпечує автоматизовану допомогу в розробці ПЗ, його супроводженні, керуванні діяльністю розробників.

Прикладами реалізації технології є такі програмні засоби, як RUP/USDP (використовує UML на всіх етапах розробки ПЗ), IBM Rational Rose, Borland Together, Telelogic Tau, Microsoft Visio/UML Add-on.

Засоби моделювання

- **UML** (англ. *Unified Modeling Language* — уніфікована мова моделювання), використовується у парадигмі ООП
- **BPMN** (англ. *Business Process Model and Notation* — система умовних позначень (нотація) для моделювання бізнес-процесів)
- **Rational Rose** — світовий лідер серед засобів візуального моделювання

Технологія CASE передбачає наявність:

- UML для побудови моделі ПЗ
- графічні редактори для побудови діаграм
- генератори для генерації коду для різних платформ
- репозиторій з базою даних, у якій зберігаються результати розробки компонентів ПЗ

Запитання для перевірки знань

1. Дайте визначення поняття архітектури програмного засобу.
2. Поясніть поняття візуального моделювання.
3. Наведіть приклади програмних засобів, які можна використати для графічного подання архітектури ПЗ.
4. Що таке універсальність мови моделювання?
5. Поясніть, яка користь у використанні стандартних мов моделювання.
6. Наведіть приклади графу, в якому як сутність є учень. Перелічіть його зв'язки з іншими сутностями.

12.4. Діаграми UML. Діаграми прецедентів

Проектування програмного забезпечення (ПЗ) починається з формулювання вимог до нього.

Вимоги до ПЗ — це саме ті функції, які має виконувати дане ПЗ, його сервіси. Саме з цієї причини процес розроблення починається з оголошення й аналізу вимог, які будуть виконувати розробники, і як це буде виконуватися, знають тільки вони, а замовник оцінить кінцевий результат роботи — створене ПЗ (рис. 12.8).

Верифікація (англ. *verification* — узгодження) — це процес перевірки ПЗ або його компонентів на відповідність отриманих результатів визначеним вимогам.

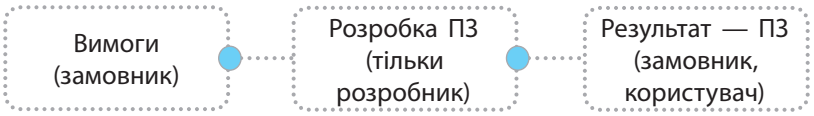


Рис. 12.8. Загальна схема розробки ПЗ

Вимоги до нового ПЗ обов'язково документуються — створюється технічне завдання (ТЗ) проекту, воно має вигляд звичайного тексту. Окремі пункти ТЗ часто подають у графічному вигляді. Набір вимог мови візуального моделювання матиме вигляд діаграми вимог (прецедентів) та набору додаткових нефункціональних вимог.

Прецеденти, або варіанти використання (*use-case*), застосовуються для опису зовнішніх вимог до нового ПЗ або вимог удосконалення та зміни вже існуючого, а також правила взаємодії користувача з ПЗ.

Нефункціональні вимоги не враховують конкретний варіант використання, у них може бути вказано особливості середовища реалізації, швидкодія, можливість додавання нових функцій та ін.

Спочатку вимоги записують у текстовий документ, який надалі оформлюють як таблицю, в одному стовпці якої будуть прецеденти, а в іншому — прізвища виконавців.

Приклад 1.

Замовили програмний засіб, при запуску якого малюється фігура. При узгодженні вимог розробник розуміє, що фігура — червоний квадрат. Здійснюється розробка ПЗ. Перевіряється результат: фігура — квадрат? Колір — червоний? Це *верифікація*. При здачі ПЗ замовнику з'ясовується, що червоний квадрат — саме те, що він очікував. Або замовник хотів рожевий прямокутник і ПЗ слід переробляти. Це *валідація*.

Приклад 2.

Складемо таблицю прецедентів вступу випускників у заклад вищої освіти, тобто таблицю вимог від учня та закладу:

Прецедент	Виконавець
Отримати атестат	Учень, працівники закладу середньої освіти
Отримати сертифікати ЗНО	Учень
Подати копії документів до вишу	Учень, працівники приймальної комісії
Оприлюднити результати подання документів	Група 1 закладу вищої освіти
Здійснити конкурсний відбір	Група 2 закладу вищої освіти
Оприлюднити список студентів	Група 1 закладу вищої освіти
Подати оригінал документів до вишу	Учень, працівники приймальної комісії
Видати довідку про вступ	Працівники закладу вищої освіти

Валідація — це визначення відповідності ПЗ очікуванням і потребам замовника (користувача).

Рис. 12.9. Вигляд ектора (*а*) та прецедента (*б*)

Таблицю можна подати й у вигляді графу, в якому вершинами (сутностями) є учні, різні працівники закладів освіти, набір прецедентів. Кожне ребро графу покаже зв'язок між прецедентами та дійовими особами. Вигляд графу і буде діаграмою прецедентів.

Опис прецедентів не вказує, як досягається певний результат, ідеться лише про загальний опис дій (як учень готується до ЗНО та здає, вчиться й отримує атестат, як здійснюється вибір майбутніх студентів, не розглядається). Тому діаграми прецедентів відносять до групи діаграм, за якими візуалізуються динамічні аспекти архітектури системи.

У діаграмі прецедентів сутності називають екторами.

Ектор (від англ. *action* — дія) — множина ролей, які виконуються в процесі взаємодії дійових осіб та прецедентів.

За правилами мови візуального моделювання ектори зображують у вигляді людини або як клас у програмуванні, а прецеденти описують у фігурі (еліпсі) (рис. 12.9).

Прецеденти й ектори з'єднують за допомогою лінії, яка також може мати стрілку — стрілка спрямована до елемента, послуги якого використовують. В описі ПЗ стрілка спрямована до елемента, у якого запитується сервіс — функція ПЗ.

Частково перетворимо таблицю на діаграму прецедентів за правилами мови візуального моделювання (рис. 12.10).

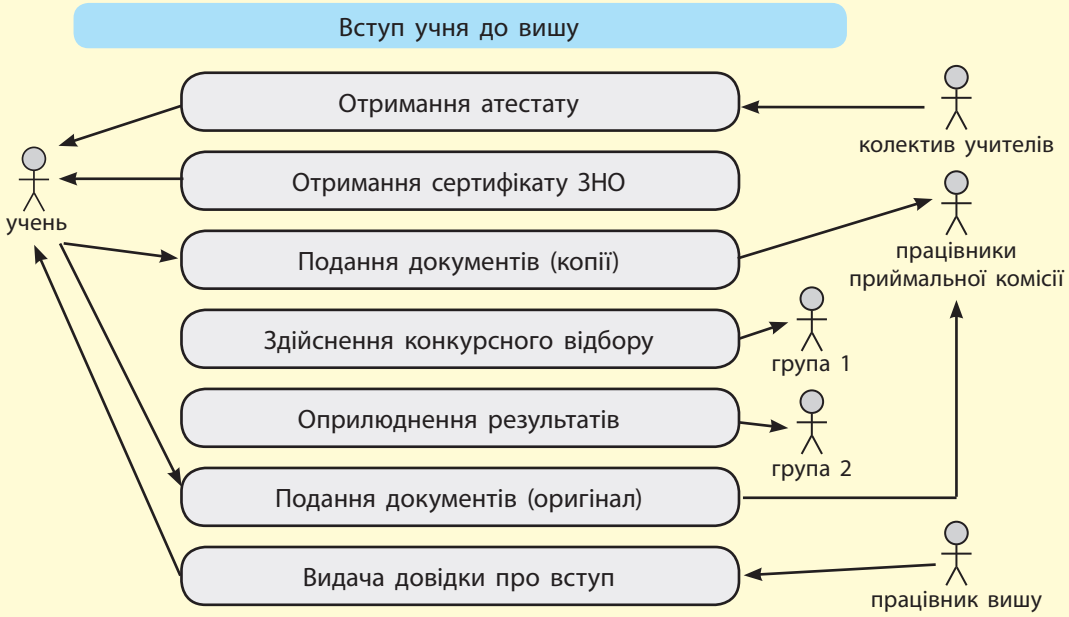


Рис. 12.10. Приклад діаграми прецедентів для завдання з організації вступу випускників до вишу

На діаграмі відсутні деякі інші вимоги до процесу вступу, наприклад учню потрібно пройти співбесіду. Цю вимогу можна додати у вигляді прецедентів, що допоможе графічно показати зв'язок сутності учень та працівників різних підрозділів вишу.

Праворуч від прецедента можна показати відгук на нього, щоб візуалізувати ланцюжок:

учень (випускник) → подання документів (копій) → працівники приймальної комісії

Це означає, що випускник здав документи, а працівник комісії їх прийняв. Такі відгуки розміщують у фігурі прямокутної форми, вони становлять основу діаграм взаємодій.

Підсумовуємо: створення діаграми прецедентів відбувається послідовно (рис. 12.11). Екторами можуть бути предмети, наприклад функцією зовнішнього e-mail-шлюзу є надсилання рекламних повідомлень, які, у свою чергу, підготовлені працівниками фірми).

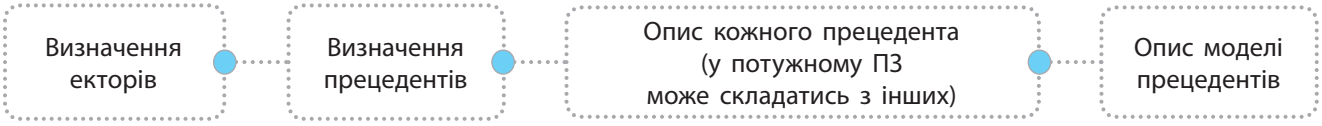


Рис. 12.11. Створення діаграми прецедентів

Діаграми прецедентів є потужним засобом подання архітектури ПЗ, засобом досягнення взаєморозуміння між замовниками, розробниками, експертами, користувачами (пригадайте, що комікси набагато легше сприймаються і дають уяву про зміст, ніж простий опис у вигляді тексту).

Оскільки ПЗ складаються з великої кількості складових, має велике число операцій (функцій окремих частин і всього ПЗ у цілому), то моделювання прецедентів дозволяє краще уявити функціонування ПЗ.

Прецеденти є основою для тестування складових під час розроблення ПЗ, адже діаграма прецедентів описує бажані вимоги до ПЗ з точки зору замовника і користувача, а в процесі постійного порівняння наявних функцій складових з описаними прецедентами відбувається контроль реалізації функцій складових.

Мовою UML пропонується така послідовність створення типів діаграм:
- діаграма прецедентів
- діаграма класів та діаграма об'єктів
- діаграма послідовностей
- діаграма станів
- діаграма взаємодій
- діаграма активностей...

Запитання для перевірки знань

1. Як ви розумієте поняття прецедента?
2. З яких елементів створюють діаграму прецедентів?
3. Яке призначення діаграми прецедентів?
4. Які аспекти розроблення ПЗ не враховує діаграма прецедентів?
5. Проаналізуйте вміст рис. 12.10 та діаграми та поясніть, які прецеденти таблиці є відгуком на прецеденти учня.
6. За аналогією створення коміксів складіть діаграму поведінки учня або учениці на уроці фізкультури.

Завдання для самостійного виконання

Уявіть себе розробником найпростішого графічного редактора, наприклад такого, як MS Paint. Екторами будуть інструменти редактора (назвіть їх у загальному вигляді, наприклад фігури, гумка, ножиці тощо). А прецедентами будуть дії, які можна здійснювати до екторів, наприклад створення, масштабування, фарбування тощо.
Створіть діаграму прецедентів для цього програмного забезпечення.

12.5. Моделювання даних і архітектури програмного забезпечення

Пригадайте визначення класів об'єктно-орієнтованого програмування (ООП), парадигми ООП і модифікаторів доступу до класу.

Як зображують ектор на діаграмі прецедентів?

Мови візуалізації спрямовані на побудову різних моделей архітектури програмного забезпечення (ПЗ) за допомогою різних діаграм (приклад 1). Пригадайте, як у ході пояснення складових діаграми прецедентів наголошувалося, що ектор можна подати як клас в ООП, адже клас — це опис сукупності об'єктів із загальними атрибутами, операціями і семантикою.

Діаграма класів (*class diagram*) — це опис сукупності статичних складових ПЗ.

Діаграма класів показує, опис яких сутностей буде опрацьовано в коді програм. Програмний інструментарій, оснований на UML-моделюванні, дозволяє автоматично перетворювати діаграму на код.

Діаграма класів є кінцевим етапом проектування, після якого розробники приступають до написання коду. Опис класів подібний до опису сутностей з їх атрибутами в проектуванні бази даних.

За правилами мови моделювання клас на діаграмі зображують у вигляді прямокутника, який розділено горизонтальними лініями на три частини, кожна з яких містить назву, атрибути та операції класу (рис. 12.12).

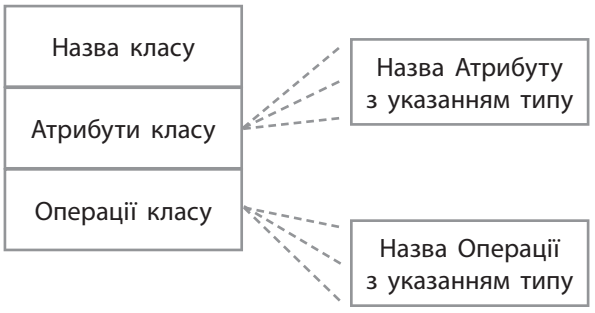


Рис. 12.12. Схематичне зображення класу в діаграмі класів

Як відбувається опрацювання класу, описано в загальному вигляді, як це здійснено на рівні кодів — знає розробник коду (а користувачів ПЗ це питання не цікавить). Тут працює відома вам парадигма ООП — інкапсуляція. Доступ до атрибутів і операцій класу здійснюється завдяки описаним модифікаторам доступу до класу (приклад 2).

У побудові діаграм ураховують і інші парадигми ООП.

Приклад 1. Одна з діаграм описує взаємодію користувача з ПЗ, інша — зміну його станів у процесі її роботи, ще одна — взаємодію складових (компонентів) ПЗ.

Приклад 2. Вам необхідно надіслати СМС-повідомлення за допомогою мобільного телефону. Ви переходите до вибору адресата й вибираєте значок повідомлення, далі вводите текст і відправляєте повідомлення. Реалізація операцій відбувається у вікні, яке представляє інтерфейс об'єкта адресат.

Об'єкт може мати кілька інтерфейсів: в адресата є редагування його даних, надсилання повідомлень, надсилання електронного листа тощо.

Якщо взяти як приклад графічний редактор, то фігура матиме інтерфейс зафарбування, копіювання, переміщення тощо.

Наслідування реалізується в створенні суперкласів (класів-батьків) і підкласів (класів-нащадків) (рис. 12.13), а поліморфізм у перевизначенні операцій — методів.

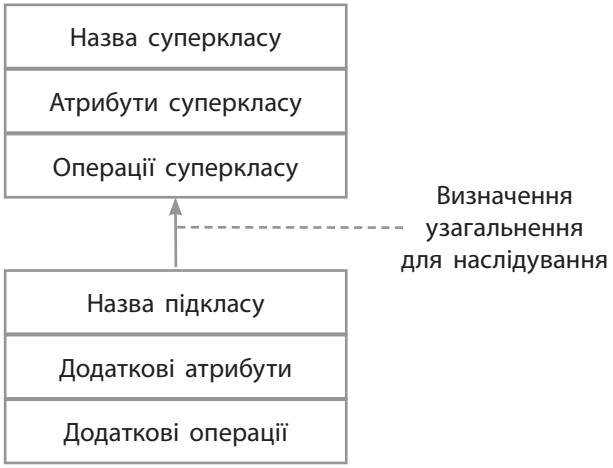


Рис. 12.13. Схематичне зображення властивості наслідування в діаграмі

Продовжимо аналізувати приклад з графічним редактором.

Суперклас Фігури матиме атрибути координат x і y розташування центра фігур, для підкласу точки додаткових атрибутів не потрібні, а прямокутник матиме ще й атрибути висоти та ширини, круг — радіус.

У UML назву класу записують з великої літери, перед назвою атрибуту або операції вказують позначку, яка відповідає певному модифікатору доступу: для private — це _, для protected — #, для public — +.

Атрибут (або властивість) записують за правилами синтаксису мови ООП: вказують модифікатор видимості, тип, ім'я з малої літери (приклад 3).

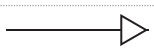
Операціями класу в ПЗ є сервіси, які надаються класом за вимогами користувачів або іншими складовими ПЗ (приклад 4).

Екземпляром класу є об'єкт. Якщо клас не утримує об'єкт, його називають абстрактним класом, а його назву записують курсивом. Також створюють і діаграми об'єктів: об'єкт на діаграмі графічно поданий так само, як і клас: назва об'єкта з класом, якому він належить, та його атрибути (приклад 5).

Діаграма об'єктів візуалізує екземпляри класу — об'єкти, їх стан і взаємозв'язок між ними в конкретний момент часу.

На діаграмі класів зображують зв'язки між класами та об'єктами класів. У UML чотири типи зв'язку: Залежність, Асоціація, Узагальнення, Реалізація.

Тип зв'язку візуалізується формою лінії та стрілки:

	Залежність
	Асоціація
	Узагальнення
	Реалізація

Завдяки поліморфізму відбувається реалізація механізму інтерфейсів. Наприклад, щойно користувач звертається до операції масштабування фігури в графічному редакторі через інтерфейс редагування, визначається клас об'єкта і викликається відповідна операція класу.

Приклад 3.
public: колірФарбування Color = (255, 0, 0).

Приклад 4.
Приклад операцій класу:
+ намалювати (фігура Прямокутник = квадрат, колірФарбування Color = (255,0,0))

Приклад 5.
Формат опису назви:
Ім'я об'єкта : клас, якому він належить (з підкресленням), наприклад:
прямокутник: Фігура.

Рис. 12.14. Зв'язки між класами на діаграмі класів

Розглянемо зазначені типи зв'язків (рис. 12.14).

Залежність (використання) — зв'язок, при якому зміни в описі одного класу впливають на інші класи. Один із цих класів буде незалежним, а інший — залежним, стрілка на діаграмі спрямована до залежного класу.

Асоціація — набір зв'язків між класами. Стрілка лінії зв'язку Асоціація спрямована на клас, який визначає правила. Якщо звернутися до прикладу БД, у якій є сутності вчитель і учень, то Асоціація навчає, а на діаграмі стрілка буде спрямована від учителя до учня з написом «навчає», для Асоціації «навчає» — спрямована від учителя до назви класу, в якому він є класним керівником. Асоціації можуть бути подвійними, тоді вони зображуються просто лінією без стрілки.

Узагальнення — зв'язок, який реалізує властивість ООП спадковість. На діаграмі він позначається лінією із замкненою стрілкою, що вказує на батьківський клас.

Реалізація — зв'язок, який відображає взаємодію між класами. Один клас визначає правила, яких має дотримуватись інший клас. Взагалі реалізація показує зв'язки між інтерфейсами та класами, які реалізують ці інтерфейси. У цьому зв'язку взаємозалежність зв'язків Узагальнення та Залежності.

Розглянемо більш детально програмування графічного редактора на прикладі 6.

Приклад 6.

Почнемо з найменшого елемента графічного зображення. Точка має параметри — координати x і y. Ці параметри встановлюються вказівником для миші. Тому точка буде мати метод фіксації положення вказівника — взяти координати та метод встановлення цих координат для точки.

Точку можна переміщувати, тому клас Точка матиме метод переміщення з параметрами-координатами, адже переміщення буде визначатися новими координатами (рис. 12.15).

Класи графічного редактора:
- Фігура
- Елемент фігури
- Точка
- Лінія

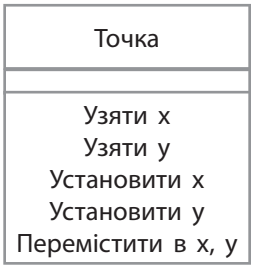


Рис. 12.15. Опис класу **Точка**

Аналогічно створюється клас Лінія.

Проте в цьому випадку встановлюється не кожна окрема координата точок її кінців, а відразу використовується точка як об'єкт, і клас Відрізок опише такі методи:
- Взяти Точку1 (вона вже має координати)
- Взяти Точку 2
- Установити Точку 1 (вже описано, як встановлюють координати точки)
- Установити Точку 2
- Перемістити в Точку (йдеться про переміщення вузла лінії як точки, що вже описано)

Елементи фігури складаються з точок і ліній, а зображення фігури створюється з елементів. У такій організації класів графічного редактора будуть задіяні два типи взаємозв'язків: Асоціація та Узагальнення.

Фігура визначається елементом — це Асоціація; стрілка спрямована від фігури. А елементи є нащадками точок і ліній (рис. 12.16).

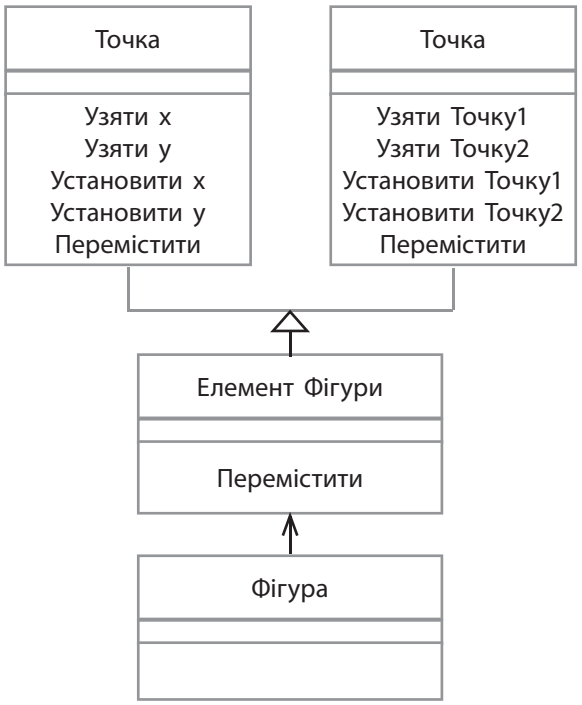


Рис. 12.16. Приклад діаграми класів для графічного редактора

У ході створення діаграми класів не завжди доцільно створювати необхідні класи, оскільки їх можна взяти з уже наявних проектів інших систем та адаптувати під своє програмне забезпечення.

Діаграма класів містить і елементи, які візуалізують інтерфейс (рис. 12.17). Після **<<interface>>** зазначають його ім'я, яке за аналогією з мовою **Java** починається з літери **I**. У мові **C++** реалізація інтерфейсу відсутня, його реалізують як перелік методів, а зміст методів реалізовано в конкретних класах

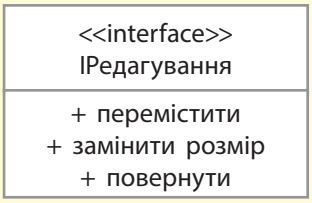


Рис. 12.17. Зображення інтерфейсу на діаграмі класів

Запитання для перевірки знань

1. Що таке діаграма класів?
2. З яких елементів на діаграмі створюють зображення класу?
3. Які типи зв'язків існують у UML між об'єктами класів?
4. Наведіть приклади реалізації на діаграмі властивості ООП наслідування, поліморфізм.
5. Відредагуйте діаграму (див. рис. 12.8), укажіть правильну назву об'єктів малювання по відношенню до класу.
6. Опишить, як показати на діаграмі редагування графічної фігури за допомогою таких операцій, як видалити, копіювати, вирізати.

Завдання для самостійного виконання

1. Доопрацюйте діаграму класів, наведену на рис. 12.16. З'ясуйте, який зв'язок між об'єктами **Точка**, **Відрізок** і накресліть його. Додайте атрибути у вказані методи. Додайте методи у клас Фігура.
2. Розгляньте діаграму класів на рис. 12.16. Вона дає уявлення про створення графічного зображення взагалі. Створіть частину діаграми побудови будь-якого об'єкта пофарбованого векторного зображення.

12.6. Діаграми послідовностей та діяльностей

Пригадайте особливості діаграми прецедентів. Які вона має складові?

Діаграми класів і об'єктів моделюють статичний стан системи програмного забезпечення (ПЗ). Взаємодія об'єктів є динамічною, якщо проявляється в передачі та прийомі повідомлень. У проектуванні ПЗ повідомленнями є виклик методів і результат їх відпрацювання.

Для графічного зображення динамічної взаємодії об'єктів користуються діаграмами послідовностей (*sequence diagram*), уточнюють діаграми прецедентів.

Діаграма послідовностей відображає взаємодію об'єктів, упорядкованих за часом.

На діаграмах послідовностей об'єкти позначено прямокутниками з підкресленими іменами (щоб відрізнити їх від класів), повідомлення — лініями зі стрілками. Суцільними лініями показано виклики методів, а пунктирними — результати відпрацювання методів.

Пригадайте, як ми складали таблицю прецедентів вступу випускників до закладу вищої освіти (§ 12.4). Створимо тепер діаграму послідовностей (рис. 12.18).

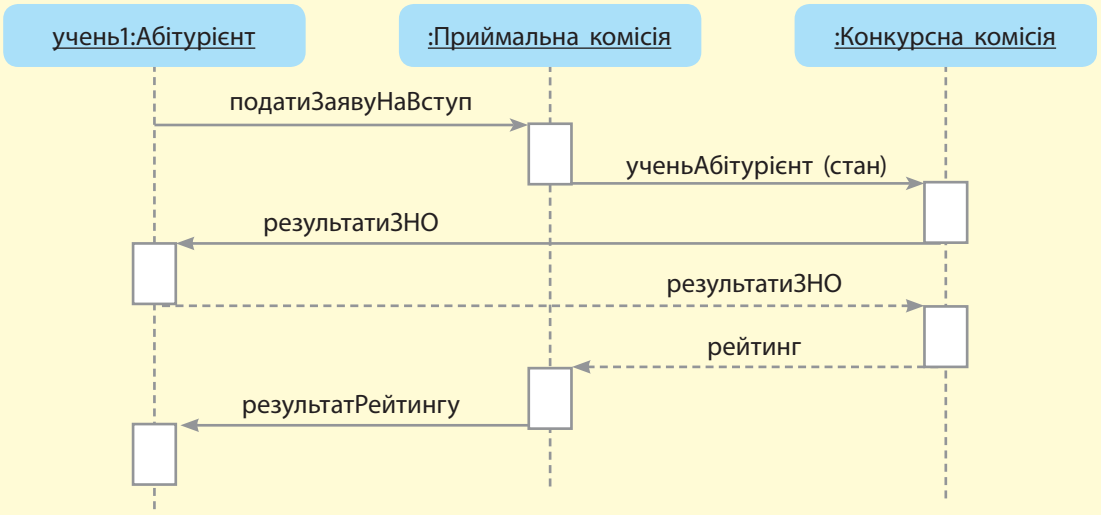


Рис. 12.18. Діаграма послідовностей вступу учнів до закладу вищої освіти

Зміст діаграми у словесній формі можна подати так: учень хоче стати студентом певного факультету. З цією метою він подає заяву в приймальну комісію, а та приймає від нього заяву, учень змінює стан на абітурієнта, конкурсна комісія перевіряє кількість балів і запитує результати ЗНО, приймає, визначає рейтинг учня серед абітурієнтів факультету, передає їх до приймальної комісії. Учень отримує результат з приймальної комісії.

На діаграмі маленькі прямокутники на вертикальних пунктирних лініях під кожним об'єктом наочно показують часову послідовність їх взаємодії (їх можна не малювати).

Альтернативою діаграми послідовностей є діаграма взаємодій. На ній зазначається не назва операції, а її порядковий номер. Оскільки від одного об'єкта може викликатися кілька методів, а між викликами отримувались результати, то в розташуванні номерів за часом не буде впорядкованості.

На рис. 12.19 подано заготовку діаграми взаємодій: показано, як у процесі створення графічного зображення користувач зафарбовує фігуру. Замість методів і їх результатів у діаграмі взаємодій проставляється нумерація в послідовності виконання.

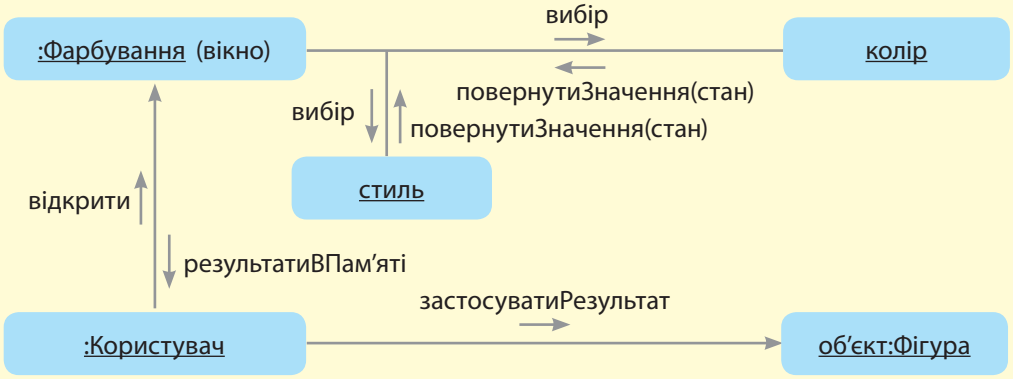


Рис. 12.19. Заготовка для діаграми взаємодій

Часто для створення алгоритму розв'язування задачі ви користуєтеся блок-схемами. У моделюванні блок-схема є найпростішим прикладом діаграми діяльностей.

Діаграми діяльностей (*Activity Diagrams*) є поданням алгоритмів процесів, які виконуються в системі. Елементами такої діаграми є операції.

Діаграми діяльностей уточнюють, деталізують особливість алгоритмічної і логічної реалізації операцій, візуалізують послідовність етапів складного процесу, наприклад обчислювального алгоритму або технологічного процесу.

Для певного ПЗ діаграм діяльностей може бути кілька, і в кожній демонструється перехід однієї послідовності дій до іншої. На відміну від блок-схем, на яких одинарними стрілками показується послідовний перехід між алгоритмічними структурами, на діаграмі зображують паралельний потік дій між елементами.

На рис. 12.20 показано, як зображуються операції до певного елемента діаграми, які змінюють його стан. У випадку *а* прикладом є реалізація розгалуження, у випадку *б* показано об'єднання операцій, застосованих до певного елемента. Червоним (випадок *в*) позначено недопустимий варіант — заборона паралельних операцій до об'єкта та від нього.

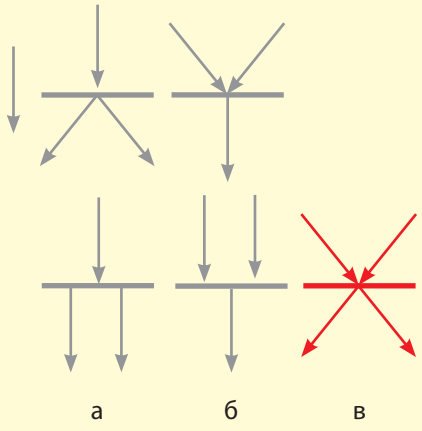


Рис. 12.20. Вигляд потоків дій на діаграмі діяльностей

У нотаціях UML паралельний потік дій називається *смугами* (swimlanes) і є частиною області діаграми тільки тих дій, за які відповідає цей елемент. Діаграми діяльностей із смужками мають вигляд басейну з плавальними смугами.

Кожна зі смуг відповідає за певний елемент та в області яких показано перелік дій з елементом.

Основне завдання діаграми діяльностей — показати послідовність етапів складного процесу та деякі обмеження, накладені на ці етапи. Розглянемо діаграму діяльностей на прикладі вступу абітурієнта до вишу. Після подання документів особі цікаво, яка вона в рейтингу вступників. Абітурієнт виходить на сервер за відомим йому посиланням для перегляду списків і свого рейтингу (рис. 12.21).

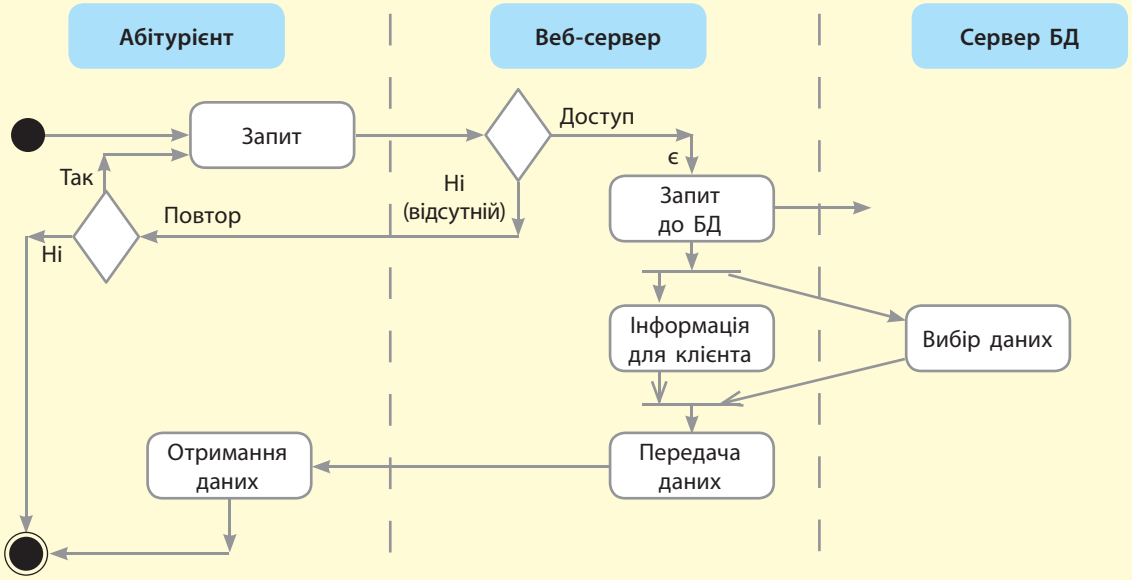


Рис. 12.21. Приклад діаграми діяльностей

На діаграмі вертикальними смугами зображено смуги з іменами Абітурієнт, Веб-сервер, Сервер БД. Їх можна й не малювати, також смуги можуть бути горизонтальними — все зображення необхідно розвернути на 90 градусів. Початкова точка діаграми (або початковий стан об'єкта) — чорний круг, а кінцева — чорний кружечок у білому кружочку. Стрілками зображено напрями потоків даних, у закруглених прямокутниках — назву операції.

Запитання для перевірки знань

1. Для чого створюють діаграми послідовностей?
2. Які складові відображено на діаграмі послідовностей?
3. Яка різниця між діаграмами послідовностей і взаємодій?
4. Що моделює діаграма діяльностей?
5. Яка відмінність діаграми взаємодій від діаграми активностей?
6. Як показати на діаграмах процес редагування графічної фігури з операціями: видалити, копіювати, вирізати?

Завдання для самостійного виконання

Проаналізуйте приклад, наведений на рис. 12.21, та створіть діаграму взаємодій з нумерацією повідомлень. Додайте на свій розсуд додаткові дії.

12.7. Проектування інтерфейсу користувача

Пригадайте, що таке інтерфейс. Які типи інтерфейсів ви знаєте?

Користувач або замовник працює з програмним засобом через інтерфейс, отже, за його зручністю й функціональним призначенням буде надано оцінку працездатності кінцевому продукту роботи розробників.

Якщо завданням є створення нового сайта для просування й замовлення певної продукції, то оформлення сайта, врахування в його інтерфейсі вимог різного типу клієнтів відіграватиме значну роль у розповсюдженні та продажі продукції. Зручне оформлення графічного інтерфейсу є вирішенням проблем його графічного дизайну.

Інтерфейс програмного засобу має багато призначень, його називають також інтерфейсом прикладного програмування.

Інтерфейс прикладного програмування (від англ. *Application Programming Interface, API*) — це комплекс готових класів, методів, структур, констант, які пропонуються додатком (бібліотекою, сервісом) або операційною системою користувачу або іншим програмним засобам.

Оформлення інтерфейсу, його дизайн залежить від того, для кого призначено програмний засіб, а також від функціонального наповнення засобу.

Інтерфейс користувача (англ. *user interface*, *UI*), або користувацький інтерфейс, призначено для зручної та комфортної роботи з програмним засобом. Правильно спроектований інтерфейс називають *дружнім*, або *інтуїтивно зрозумілим*.

Розробка інтерфейсу користувача складається з таких етапів, як дослідження, проектування структури, проектування дизайну, безпосереднього розроблення. Розглянемо їх.

Сучасний вигляд інтерфесу **SILK-інтерфейс** (від англ. *Speech* — мова, промова, *Image* — зображення, образ, *Language* — мова, *Knowledge* — знання) дозволяє керувати програмою на основі поведінки користувача. В проектуванні таких інтерфейсів використовувалися мовні та біометричні технології: команди можна давати, промовляючи їх, а в деяких іграх можна керувати персонажем через рух користувача.

Етап 1	На **етапі дослідження** відбувається аналіз призначення ПЗ, статистики використання подібного типу ПЗ, пристроїв для розміщення розроблюваного ПЗ, функціональних особливостей ПЗ, які мають бути представлені користувачу, створення списку задач, які будуть реалізовані в інтерфейсі користувача. Етап дослідження є основою для етапу проектування, на якому створюється структура інтерфейсу, оскільки завдання, реалізовані в інтерфейсі, складаються з кроків їх розв'язування, то представлений інтерфейс ПЗ може мати кілька екранів для уможливлення виконання всіх вимог до ПЗ.
Етап 2	На **етапі проектування структури інтерфейсу** створюють *прототип інтерфейсу* — зазвичай два: чорновий та фінальний. У *чорновому* створюють схему інтерфейсу із зонами для груп елементів без їх детального опису. Чернетка дозволяє з'ясувати, наскільки складним буде інтерфейс, чи достатньо одного екрана для реалізації всіх вимог до ПЗ. *Фінальний прототип* містить опис елементів у зонах з їхнім призначенням і зв'язками з іншими елементами. Цей етап часто завершується створенням діаграми взаємодій.

Етап 3	**Етап проектування дизайну інтерфейсу** розв'язує питання оформлення інтерфейсу (кольорова гама, розмір і вигляд елементів, їхнє взаємне розташування, наявність графічних і анімаційних ефектів). Після створення концепції дизайну є уявлення, яким буде інтерфейс ПЗ.
Етап 4	**Етап безпосереднього розроблення інтерфейсу** передбачає наявність макетів усіх екранів інтерфейсу, необхідних матеріалів та документації дають можливість розробникам перейти до наступного етапу.

Розглянемо деякі вимоги до дизайну інтерфейсу.

- Загальний вигляд інтерфейсу визначається взаємним розташуванням його елементів. Зазвичай елементи групують за призначенням.
- Для визначення розмірів груп елементів користуються правилами пропорції (наприклад, принципом «золотого перетину»). Кожний елемент має своє призначення, про що є підказка. Зручно, коли вона з'являється у спливаючому віконці. Такі підказки не виключають наявності довідки щодо використання ПЗ. Інтерфейс має бути інтуїтивно зрозумілим, бо часте звертання до довідки уповільнює роботу.
- В інтерфейсі програмних засобів для колективного користування групою слід обережно розміщувати (а інколи є сенс і приховати) команди особистого налагодження засобу. Працівники звикають до певного вигляду інтерфейсу, і щоразу пристосовуватися до змін, уведених попереднім користувачем програми, не зручно.

Правила оформлення інтерфейсу користувача
- Групування елементів (меню, форми, блоки)
- Вирівнювання елементів
- Єдиний стиль елементів за формою та кольором
- Наявність вільного простору для розмежування блоків

У 1990 році дослідники Якоб Нільсен (на фото зліва) і Рольф Молич оголосили 10 евристик — правила, що найчастіше використовуються в проектуванні дизайну інтерфейсу.

Описані етапи розробки інтерфейсу складають основу дизайну інтерфейсу ПЗ, в якому поєднано:
- UI-дизайн (від. англ. *User Interface* — користувацький інтерфейс);
- UX-дизайн (від. англ. *User Experience* — досвід взаємодії);
- продуктовий дизайн.

Користувацький інтерфейс (інтерфейс, призначений для користувача) є більш вузьким поняттям у визначенні інтерфейсу. Як вже наголошувалося, користувацький інтерфейс складається з певного комплексу графічно оформлених технічних елементів (кнопки, селектори та інші поля). Його призначенням є допомога користувачеві в організації зручної взаємодії з програмою або сайтом.

Проте призначення інтерфейсу цим не обмежуються: в інтерфейсі також мають бути реалізовані завдання програмного засобу. Якщо створюється інтерфейс сайта певних послуг, то враховуються запити користувача до цих послуг, зручність у їх виклику, наявність довідкових матеріалів до послуг, організцій, що забезпечують послуги, взаємодія з ними тощо.

Якщо створюють новий програмний засіб, наприклад графічний редактор, то при його запуску користувачеві забезпечуються зручні умови його використання та виклик функцій для роботи з ним. Інтерфейси графічних редакторів враховують також досвід користувача: так, стандартний, убудований в операційну систему MS Windows растровий графічний редактор Paint розрахований на менш досвідченого користувача,

ніж інші потужні редактори, які передбачають певний досвід для їх повнофункціонального використання.

UX-дизайн враховує компоненти взаємодії «користувач — ПЗ»: інформаційну архітектуру, проектування взаємодії, інтерактівність користувацького інтерфейсу, його наповнення.

Продуктовий дизайн — це весь процес створення інтерфейсу, який починається з аналізу розв'язання проблеми представлення ПЗ користувачеві або замовнику до розробки плану тестування інтерфейсу на відповідність вимогам та оформлення декількох його концепцій.

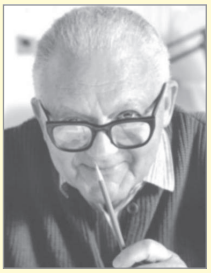
Фахівці з розробки продуктового дизайну називаються продуктовими дизайнерами (*Product Designer*) — цей фах з'явився в новому тисячолітті (після 2000 року).

Зазвичай продуктовим дизайном опікується група фахівців, які підтримують зв'язок із розробниками впродовж часу створення інтерфейсу. У сфері просування продукції продуктові дизайнери також взаємодіють із працівниками відділу маркетингу. Великі компанії ми пізнаємо за фірмовим логотипом. Зазвичай логотип є обов'язковим елементом дизайну сайта компанії чи закладу. Пол Ренд сформулював тестові запитання, відповіді на які однозначно окреслять логотип компанії. За тестами славетного дизайнера й сьогодні визначають ефективність використання логотипів.

Зупинимось на кількох принципах проектування.

При створенні дизайну інтерфейсу на основі правил Золотого перетину користуються співвідношенням 1 : 1.618 для розмірів та взаємного розташування елементів інтерфейсу. Поєднання симетрії і правила золотого перетину сприяє найкращому зоровому сприйняттю і появі відчуття краси і гармонії. Часто на основі правила золотого перетину створюють розмітку веб-сторінок. Так, логотип Apple (надкушене яблуко) побудований за принципом золотого перетину.

Гаманець Міллера (закономірність виявлена американським вченим-психологом Джорджем Міллером) полягає в тому, що короткочасна людська пам'ять може запам'ятати і повторити лише 7 (± 2) елементів. Цей принцип поширено використовується в побудові інтерфейсів програм. Якщо кількість елементів меню (пунктів меню, кнопок, закладок) більше від 7 (або насамкінець 9), то їх намагаються згрупувати.



Американського дизайнера і арт-директора Пола Ренда називають легендою епохи. Він є автором розробки логотипів відомих фірм **IBM**, **UPS**, **Enron**, **Morningstar**, **Inc**, **Westinghouse**, **ABC**, а також логотипа **NeXT**, розробленого на замовлення Стіва Джобса.

Принципи проектування дизайну інтерфейсів

- Золотий перетин
- Гаманець Міллера
- Принцип групування
- Бритва Оккама або KISS
- Видимість відображає корисність
- Розумне запозичення

Запитання для перевірки знань

1. Назвіть етапи розроблення інтерфейсу.
2. Дайте визначення поняттю продуктового дизайну.
3. Які проблеми розв'язує група продуктових дизайнерів?
4. До якого етапу розробки інтерфейсу графічного редактора можна віднести створення діаграми взаємодій для нового зображення, що створюється з «0» або завантажується з файла?
5. Чим відрізняються обов'язки дизайнера і продуктового дизайнера?
6. Створіть макети інтерфейсу вікон графічного редактора.

12.8. Тестування та оцінювання програмного забезпечення

Пригадайте етапи розробки інтерфейсу ПЗ. Яке призначення прототипів на етапі проектування структури?

Для створення інтерфейсу ПЗ розробники використовують прототипи інтерфейсу. Реалізація кожного етапу розробки ПЗ теж базується на створенні прототипів, тому прототипи діляться на типи. Ви вже дізналися про інтерфейсні прототипи (їх ще називають горизонтальними прототипами).

Горизонтальний прототип демонструє функціональні можливості ПЗ (але не забезпечує їх працездатність): показує, як виглядає користувацький інтерфейс (кольори, графічні елементи, вікна та елементи управління) і визначає структуру навігації.

Тестування ПЗ — це процес перевірки, наскільки ПЗ відповідає вимогам і очікуванням. Розроблені для цього тести мають основою задокументовані вимоги замовника.

Горизонтальним прототип називають тому, що в ньому не передбачено реалізацію видів архітектури, функції системи, натомість у ньому втілюються особливості інтерфейсу користувача.

У такому прототипі під час виконання переміщення між об'ктами зазначатиметься, що саме має знаходитися в місці переміщення, оскільки здійснення функціональних операцій є особливостями іншого типу прототипів.

Вертикальний (або структурний) прототип не обмежується інтерфейсом користувача. У ньому враховано всі етапи розроблення ПЗ. Вертикальний прототип дозволяє перевіряти правильність архітектурних рішень.

Прототипи створюють для перевірки працездатності ПЗ: на початку розроблення вимог до ПЗ не завжди є можливість встановити однозначність цих вимог. Крім того, вже в процесі розроблення можуть виникнути доповнення до вимог. Для з'ясування відповідності вимогам розрізняють одноразові (або дослідницькі) та еволюційні прототипи.

Одноразові прототипи створюють для отримання відповіді на певне питання щодо ПЗ, для покращення певних вимог до програмного засобу. *Еволюційні прототипи* вважають архітектурним «фундаментом» для поступового створення кінцевого результату з урахуванням появи нових вимог і з'ясування нюансів існуючих.

Створення еволюційного прототипу є одним з компонентів моделі спірального циклу розробки ПЗ: розроблюється прототип початкової версії ПЗ, який надалі модифікується до остаточного результату, який відповідає меті і вимогам проекту (рис. 12.22).

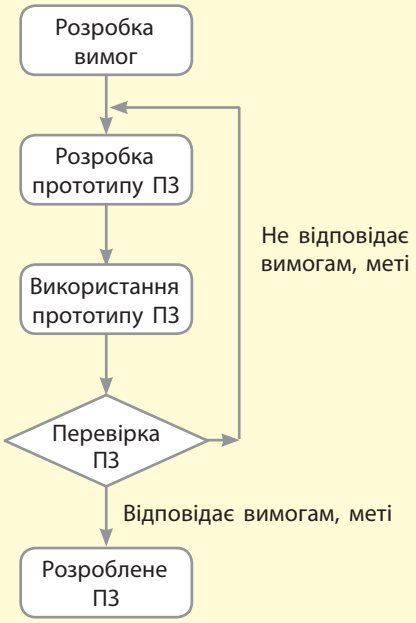


Рис. 12. 22. Схема використання еволюційного прототипу

Створення прототипу — це процес, який включає створення чернеток, їхню демонстрацію та критичне обговорення.

Прототипом певних етапів може бути і код, і написи на папері (дошці). Доопрацювання прототипу відбувається на основі тестування системи або її частини на відповідність вимогам.

На кожному етапі розроблення ПЗ створюють прототипи.

Прототип вимог до ПЗ — це часткова реалізація програмного забезпечення, створена з метою допомогти розробникам, користувачам і клієнтам краще зрозуміти вимоги до системи та з'ясувати питання, наскільки можливо здійснити задоволення вимог та розробити ПЗ взагалі.

Програмні прототипи є частковими реалізаціями ПЗ, для демонстрації та перевірки певних його функціональних можливостей: модульне, інтеграційне і системне тестування.

Модульним тестуванням називають перевірку працездатності окремих модулів системи. Якщо модулі об'єднують в окремі компоненти ПЗ, здійснюється *інтегроване* (або *об'єднане*) тестування. На цьому етапі також тестують користувацький інтерфейс. Наприкінці, коли вже всі модулі об'єднані та взаємодіють як єдина система, здійснюється *системне* тестування.

Уведене в експлуатацію ПЗ також підлягає тестуванню, оскільки воно використовується користувачами в абсолютно різних середовищах. Можуть виникнути помилки, пов'язані саме із взаємодією ПЗ із системним і апаратним середовищем.

Часто тестування на основі прототипів відповідає V-моделі, яку ще називають прототипним розробленням через тестування. Свою назву V-модель отримала завдяки формі її графічного подання (рис. 12.23).

Процес тестування ПЗ пов'язаний з безпосереднім розробленням ПЗ і впливає на його життєвий цикл. Життєвий цикл розробки ПЗ — це період від замовлення ПЗ до його кінцевої реалізації з впровадженням, експлуатацією та супроводом.

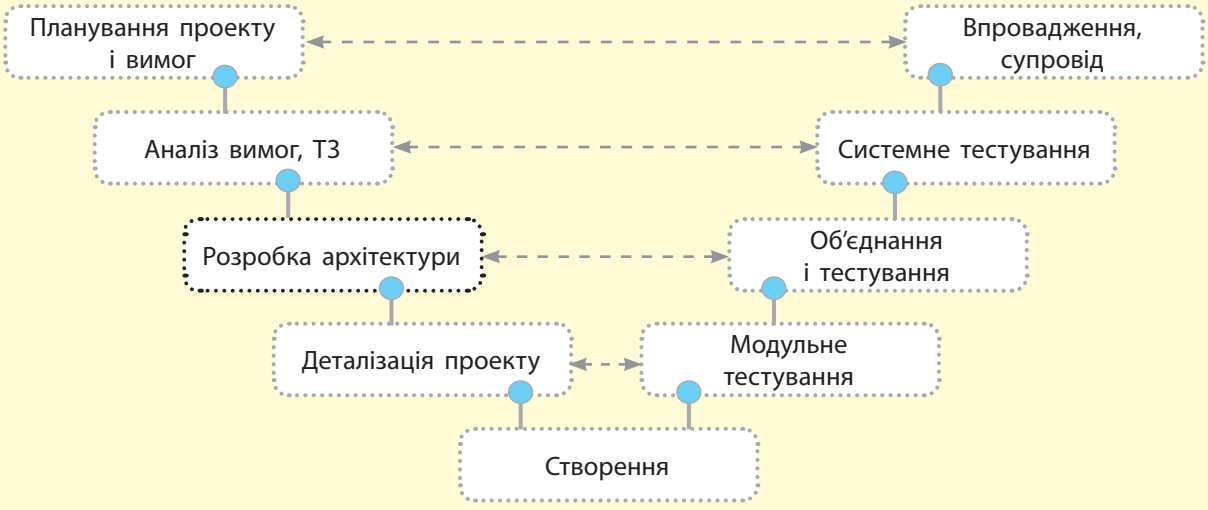


Рис. 12.23. Схема **V-моделі**

Розробка ПЗ також може здійснюватися на основі спіральної моделі з використанням прототипів. Така розробка є циклічним процесом, у якому відбувається багаторазове повторення процесів створення прототипів, тестування та коректування проектування, доки не буде досягнуто мети розробки.

План тестування — це документ, в якому описано весь обсяг робіт із тестування: опис ПЗ, стратегія, розклад, критерії початку та закінчення тестування, а також необхідне обладнання, спеціальні знання та оцінки ризиків з варіантами їхнього розв'язання.

Результати роботи з прототипом дозволяють своєчасно коригувати вимоги до ПЗ. Після цього перед розробниками постають лише перевірені й деталізовані завдання.

Часто в описі процесів тестування зустрічаються абревіатури QA і QC.
QA (від англ. *Quality Assurance* — гарантія якості) означає процес забезпечення якості, який охоплює всі етапи розробки програмного засобу. QA передбачає вивчення процесів і визначення всіх умов і обставин, які можуть вплинути на якість розробки та її кінцевий результат.
QC (від англ. *Quality Control*) означає контроль якості ПЗ при розробці та його аналіз на відповідність вимогам на певному етапі.

Тест-дизайн — це етап тестування ПЗ, на якому проектуються та створюються тести (тест-кейси) відповідно до визначених критеріїв якості ПЗ та мети тестування.

У процесі здійснення тест-дизайну тест-аналітик визначає, ЩО тестувати, а тест-дизайнер — ЯК тестувати.

Існують різні методи тестування. Розглянемо деякі з них.

Еквівалентний розподіл — це перевірка правильної роботи програмного засобу для допустимого діапазону значень вхідних даних.

Наприклад: є діапазон значень певних параметрів ПЗ (від 14 до 60) — під час тестування перевіряється робота ПЗ для значень діапазону та для значень, за його межами. Або під час тестування програми з визначення кореня квадратного рівняння необхідно вводити значення параметрів, при яких є розв'язок та при яких він відсутній.

Аналіз граничних значень — для наведеного вище прикладу беруться значення межі 14 та 60, а також значення, менші за 14 та більші від 60.

Причина/Наслідок — під час проектування системи керування базою даних необхідна функція додавання в таблицю даних: здійснюється перевірка: викликається функція додавання, вводиться значення, підтверджується додавання (причина) та перевіряється, чи присутнє в таблиці нове значення (наслідок).

Передбачення помилки — при введенні нового номера телефону в базу даних перевіряється кількість цифр і видається повідомлення в разі неправильної кількості.

Вичерпне тестування — відбувається введення комбінацій значень всіх вхідних даних, не завжди використовується у випадку великої кількості вхідних даних.

Попарне тестування — формування наборів вхідних даних, при якому значення одного параметру хоч би раз використовувалось із значенням кожного іншого параметру.

У процесі тестування визначається якість ПЗ.

Якість програмного забезпечення (*Software Quality*) визначається сукупністю його характеристик, які відповідають установленим вимогам і очікуваним потребам.

Це процес оцінювання системи або її компонентів. Він здійснюється з метою визначення, чи задовольняють результати етапів розробки, сформованих на початку їх виконання, такі як мета, термін завдання кожного етапу.

Запитання для перевірки знань

1. З якою метою розробляють прототипи?
2. Для чого розробляють прототип вимог до ПЗ?
3. Які завдання розв'язують вертикальні та горизонтальні прототипи?
4. Яка різниця в розробці ПЗ на основі V-моделі та спіральної моделі?
5. Як здійснюється тестування ПЗ?
6. З чого складається оцінювання ПЗ?

12.9. Системна архітектура. Апаратні та програмні рішення

Що таке архітектура ПЗ?

Замовити програмне забезпечення (ПЗ) можуть організації, що мають свою структуру, обчислювальні пристрої, мережу, програмні засоби. Сучасні організації застосовують інформаційні технології для вдосконалення методів роботи.

Ознайомимося з ПЗ, у якому враховано бізнес-діяльність організації, її зв'язки із зовнішніми структурами, а також яке задовольняє інформаційні потреби всіх співробітників, служб і підрозділів. Таке ПЗ називають інформаційною системою (ІС).

Системною називають архітектуру, яка визначає сукупність методологічних, технологічних і технічних рішень для забезпечення інформаційної підтримки діяльності організації.

Системна архітектура визначає загальний склад ІС організації й зазвичай має такі складові: архітектуру додатків, архітектуру даних і архітектуру обладнання.

Архітектуру додатків становлять функціональний і компонентний склад ІС. До її складу належать додатки, які підтримують виконання бізнес-процесів, інтерфейси взаємодії підсистем та ІС із зовнішніми системами, засоби і методи розробки та супроводу додатків.

Архітектурою даних є методи взаємодії складових системи і зберігання даних, безпосередньо сховища даних і правила доступу до них.

Архітектуру обладнання складають мережна архітектура, програмні та апаратні засоби обчислювальної техніки.

Як ви знаєте, для опису різних моделей архітектури використовують графічні схеми, мови графічного моделювання, правила яких розроблені на основі стандартів.

IDEF є сукупністю стандартів моделювання та проектування ІС. Кожна складова цієї сукупності описує окрему модель архітектури. Спершу був розроблений стандарт IDEF0, далі додавалися моделі, і на місці 0 з'явились цифри 1, 2, ...

Стандарт ARIS (*Architecture of Integrated Information System, ARIS*) є сукупністю різних методів моделювання, які відображають різні аспекти ІС. ARIS призначено для розробки ІС — від визначення вимог до опису реалізації. У цій архітектурі враховано опис організаційної структури, моделювання бізнес-процесів, документування архітектури додатків, даних тощо.

ARIS пропонує діаграму eEPC (*extended Event Driven Process Chain* — розширений ланцюжок процесів керування подіями). Вона розширює можливості сукупності стандарта IDEF

Стандарт **IDEF0** було розроблено в 1981 році під час виконання програми автоматизації промислових підприємств з назвою **ICAM** (Integrated Computer Aided Manufacturing). У назві **IDEF** враховано назву програми: ICAM DEFinition.

IDEF1 моделює інформаційні потоки, а **IDEF3** призначений для стандартизації документування процесів у системі та для моделювання процесів, у яких важливими є послідовність виконання дій і взаємозалежність між ними.

Моделювання за вказаними стандартами використовує графічні й текстові засоби. З графічних засобів — діаграми, які й описують різні погляди на архітектуру системи.

та призначена для детального опису бізнес-процесу, модель якого являє собою послідовність подій і бізнес-функцій, спрямованих на досягнення заданого результату.

Можна також назвати міжнародний стандарт ISO — стандарт ISO серії 9000, розроблений на основі стандарту BS 7750 «Специфікації систем экологічного менеджменту» (*Specification for Environmental Management Systems*) Британською організацією стандартизації. Розуміння важливості проектування архітектури організації сприяло появі стандарту ISO 15704.

Системна архітектура сучасних організацій моделюється на основі дотримання низки принципів. Розглянемо їх.

- Серверне ПЗ функціонує в корпоративній обчислювальній мережі організації.
- Клієнтське ПЗ розподілено по світу та функціонує в будь-якому мережному середовищі.
- Використовує власні інструментальні засоби розробки ІС.
- Модель ІС має модульну структуру, що забезпечує поетапну розробку й упровадження.
- Має єдиний підхід до розробки та стандартизації ПЗ.
- Має єдиний стандартний користувальницький інтерфейс.

Запитання для перевірки знань

1. Що називають системною архітектурою?
2. Назвіть основні складові системної архітектури.
3. Навіщо створювати моделі системної архітектури?
4. Поясніть призначення складових системної архітектури.
5. Як ви розумієте поняття ІС організації?
6. Для чого розробляють стандарти архітектури ІС?

Завдання для самостійного виконання

- Зверніться до пошукової системи, у мережі Інтернет знайдіть і опишить нові стандарти й тренди в розробці ПК.

Тест 12

Тестове завдання з автоматичною перевіркою результату на сайті interactive.ranok.com.ua

Комп'ютерний словник

Адаптивність — особливий підхід до розробки сайта, який дозволяє вже існуючим веб-ресурсам підлаштовуватися під екрани різних пристроїв.

Адміністрування сайта — комплекс заходів щодо підтримки чіткого функціонування сайта, його працездатності, швидкої роботи, зручності для користувача, регулярного розміщення матеріалів на його сторінках.

Алгоритм послідовного пошуку елемента в масиві — це алгоритм, що базується на прямому переборі елементів масиву.

Алгоритм сортування масиву злиттям — це алгоритм, у якому масив ділиться на дві частини і виконується сортування окремо кожної частини.

Архітектура даних — методи взаємодії складових системи і зберігання даних, безпосередньо сховища даних і правила доступу до них.

Архітектура програмного забезпечення — структура, яка складається з програмних компонентів, їхніх інтерфейсів та опису засобів об'єднання компонентів в єдину систему.

База даних — це сховище даних різного типу про об'єкти та взаємозв'язки між ними.

Валідація сайта — перевірка синтаксичних помилок, перевірка вкладеності тегів та інші критерії.

Віджети — на JavaScript пишуться різні міні-програми, які використовуються в робочому просторі і є дуже зручними.

Візуальне моделювання — метод розробки ПЗ, який для візуалізації, опису, проектування, документування архітектури ПЗ використовує графічні моделі.

Віртуальний хостинг (*virtualhosting* або *sharedhosting*) — послуга, у рамках якої користувачеві надається частина місця на диску для розміщення веб-сайтів.

Граф — це множина вершин і множина ліній, які з'єднують дві будь-які його вершини.

Динамічне програмування — це метод розв'язування задач шляхом їх розбиття на декілька однотипних підзадач, які пов'язані між собою.

Діаграма класів (*class diagram*) — опис сукупності статичних складових моделі.

Ектор (від англ. *action* — дія) — множина ролей, які виконуються в процесі взаємодії дійових осіб та прецедентів.

Елемент (*Element*) — окремий елемент HTML, можна сказати, що це основні будівельні блоки. Атрибут (Attr) — представляє атрибут елемента.

Жадібний алгоритм — це алгоритм, для якого застосовується метод розв'язування задач оптимізації, за допомогою якого на кожному етапі вибирається такий варіант, який є найкращим на цей момент.

Запит — це об'єкт бази даних, призначений для відбору з таблиці необхідних даних, введення даних, їх опрацювання і подання користувачеві у зручній формі.

Звіт — це об'єкт бази даних, призначений для відбору з БД необхідних даних і виведення їх на екран або принтер у зручному для користувача вигляді.

Каскадні таблиці стилів (англ. *Cascading Style Sheets*, або скорочено CSS) — спеціальна мова, що використовується для опису зовнішнього вигляду сторінок, написаних мовами розмітки даних.

Кодування алгоритмів — це запис алгоритму мовою програмування.

Кросбраузерність — правильна верстка сайта, за її допомогою сторінки сайта однаково відображаються в різних браузерах. Реалізація відбувається за допомогою HTML і CSS, а також різноманітних хаків, в окремих випадках — JavaScript.

Методологія — принципи, сукупність ідей, методів і засобів, які визначають підхід до розробки програмного забезпечення.

Мови візуального моделювання (або візуальні мови) — сукупність формалізованих наборів графічних символів і правил побудови з них візуальних моделей.

Модель даних — це структура даних, яка визначає порядок зв'язків між даними.

Мультимедіа — комп'ютеризована технологія, яка об'єднує роботу з усіма джерелами даних, засіб подання різних видів інформації у цифровому вигляді.

Пошукові системи — повністю автоматизовані механізми, які глибоко сканують усі задані сервери (відкриті для сканування) і збирають індекс-інформацію про те, що і де (на якій веб-сторінці) виявлено.

Прецеденти (use-case), або варіанти використання, — застосовуються для опису зовнішніх вимог до нового ПЗ або вимог удосконалення та зміни вже існуючого, також правила взаємодії користувача з ПЗ.

Продуктовий дизайн — весь процес створення інтерфейсу, який починається з аналізу розв'язання проблеми представлення програмного забезпечення користувачеві або замовнику до розроблення плану тестування інтерфейсу на відповідність вимогам та оформлення декількох його концепцій.

Просування сайта — комплекс заходів щодо збільшення відвідуваності веб-ресурсу цільовими відвідувачами.

Реляційна модель даних — це модель, у якій об'єкти і взаємозв'язки між даними представляються за допомогою відношень.

Сервер додатків — сервер, на якому створені додатки, які використовують вашу базу даних, веб-сервіс тощо.

Серверні додатки — фрагменти коду, які виконуються на стороні сервера, де використовується Java 6.

Серверні скрипти — спеціальні програмні коди, що застосовуються для реалізації інтерактивних «властивостей» сайтів.

Система керування вмістом (англ. *Content Management System*, або *CMS*) — програмне забезпечення для організації спільного процесу створення, редагування й управління контентом веб-сайтів.

Система управляння базами даних — це програма, призначена для створення структури бази даних, уведення, оновлення, пошуку та опрацювання даних.

Система числення — це сукупність правил запису чисел за допомогою символів і виконання операцій над ними.

Системна архітектура — архітектура, яка визначає сукупність методологічних, технологічних і технічних рішень для забезпечення інформаційної підтримки діяльності організації.

Структура сайта — базис, фундамент будь-якого інтернет-ресурсу.

Теги — команди мови HTML. HTML-теги — ключові слова або символи, які записуються в кутові дужки.

Транзакція — це послідовність операцій над даними, яка сприймається СУБД як єдине ціле.

Форма — це об'єкт бази даних, призначений для введення даних і відображення необхідних даних.

Хмарний хостинг (*cloudhosting* або *cloud-storage*) — послуга з розміщення файлів користувача, за якої дані зберігаються на багатьох серверах, що розподілені в мережі.

Хо́стинг (англ. *hosting*) — послуга, що включає надання дискового простору, підключення до мережі та інших ресурсів для розміщення фізичної інформації на сервері, що постійно перебуває в мережі (наприклад, Інтернет).

Цикл графу — це шлях з одної вершини графу в ту саму вершину.

Чат-бот (англ. *Chatbot*) — комп'ютерна програма, розроблена на основі нейромереж та технологій машинного навчання, яка веде розмову за допомогою слухових або текстових методів.

Шлях графу — це послідовність його ребер, які зустрічаються при переміщенні з одної вершини в іншу.

Юзабіліті (англ. *Usability* — зручність і простота використання) — підхід, покликаний зробити сайти простими у використанні для користувача, який не потребує додаткового навчання, тобто має бути зорієнтований на нього інтерфейс.

Алфавітний покажчик

А

Адаптована верстка 178
Адміністрування сайта 151
Алгоритм
— Дейстри 104
— Флойда — Уоршелла 108
— пошуку даних 81
— у глибину 100
— у ширину 102
— сортування даних 70, 78
Анімайційний ефект 190

Б

База даних 4

В

Валідація сайта 213
Веб-програмування 198
Візуальне програмування 231

Г

Гіпертекст 159
Граф 94

Д

Динамічне програмування 112
Діаграма
— взаємодій 241
— діяльностей 241
— послідовностей 240

Е

Ектор 234
Ергономіка 219

Ж

Жадібні алгоритми 118

З

Запит 11, 39–42
Звіт 11, 28

І

Інструкція 49
Інтерфейс користувача 45, 243

К

Каскадні таблиці стилів 164
Кодування алгоритмів 58
Кроусбраузерність 184

М

Медіа-запит 174
Методологія 225
Модель даних 7
Мультимедіа 193

Н

Найкоротший шлях у графі 104

О

Обчислювальна геометрія 122
Оператор 31

П

Пошукова система 222
Прецедент 233
Прототип 246

Р

Редактор коду 154
Реляційна модель даних 8

С

Сайт 139
Система керування вмістом 148
Система управління баз даних 5
Структура сайта 145

Т

Таблиця 11
Тег 159

Ф

Форма 11
Функція 90

Х

Хостинг сайта 202

Ц

Цільова аудиторія 143
Цикл графу 96

Зміст

Передмова **3**

РОЗДІЛ 1. БАЗИ ДАНИХ

1. Загальні відомості про бази даних
1.1. Поняття бази даних і системи управління базами даних 4
1.2. Поняття моделі даних 7
1.3. Основні відомості про систему управління базами даних Access 11
2. Таблиці
2.1. Створення й уведення структури таблиць 14
2.2. Ключові поля, індекси, зв'язування таблиць 19
2.3. Введення, пошук і редагування даних у таблиці 23
2.4. Сортування і фільтрування записів. Операції над таблицями 26
3. Запити
3.1. Загальні відомості про запити 29
3.2. Запити на вибірку даних 32
3.3. Запити з функціями і з полями, що обчислюються 35
3.4. Запити з параметрами. Перехресні запити 39
3.5. Запити на змінення даних 42
4. Інтерфейс користувача. Основи мови SQL. Імпорт та експорт даних
4.1. Створення інтерфейсу користувача для введения даних у базу даних 45
4.2. Основи мови запитів SQL 49
4.3. Імпорт і експорт об'єктів баз даних 51

РОЗДІЛ 2. АЛГОРИТМИ

5. Алгоритми і числа
5.1. Методи проектування і подання алгоритмів 55
5.2. Поняття про кодування і складність алгоритмів 58
5.3. Основні поняття теорії чисел 60
6. Сортування і пошук даних
6.1. Алгоритми сортування даних 70
6.2. Алгоритми пошуку даних 81
7. Обробка рядків
7.1. Основні відомості про рядки й операції над ними 88
7.2. Функції і методи опрацювання рядків 90
7.3. Приклади програм обробки рядків 92
8. Графи
8.1. Основні поняття і терміни теорії графів 94
8.2. Способи подання графів у комп'ютері 98
8.3. Пошук у глибину та ширину 100
8.4. Визначення найкоротшого шляху у графі 104
9. Динамічне програмування і жадібні алгоритми
9.1. Динамічне програмування 112
9.2. Жадібні алгоритми 118
10. Основи обчислювальної геометрії
10.1. Базові поняття 122
10.2. Операції над векторами 124
10.3. Обчислення площі многокутника 127
10.4. Побудова опуклої оболонки 131

РОЗДІЛ 3. ВЕБ-ТЕХНОЛОГІЇ

11.1. Основні тренди у веб- дизайні 135
11.2. Види сайтів та цільова аудиторія 139
11.3. Інформаційна структура сайта 145
11.4. Системи керування вмістом 148
11.5. Адміністрування сайта 151
11.6. Інструменти веб-розробника 154
11.7. Мова гіпертекстової розмітки 159
11.8. Каскадні таблиці стилів 164
11.9. Проектування та верстка веб-сторінок 168
11.10. Адаптивна верстка 176
11.11. Кросбраузерність 182
11.12. Графіка для веб-середовища 186
11.13. Анімаційні ефекти 190
11.14. Мультимедіа на веб-сторінках 193
11.15. Об'єктна модель документа 196
11.16. Веб-програмування та інтерактивні сторінки 198
11.17. Хостинг сайта 202
11.18. Веб-сервер та база даних 206
11.19. Взаємодія «клієнт — сервер» 210
11.20. Валідація сайта та збереження даних форм 213
11.21. Прикладний програмний інтерфейс 216
11.22. Правила ергономічного розміщення відомостей на веб-сторінці 219
11.23. Пошукова оптимізація та просування веб-сайтів 221

РОЗДІЛ 4. ПАРАДИГМИ ТА ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

12.1. Уніфікований процес розробки програмного забезпечення 225
12.2. Програмні інструменти для проектної роботи 228
12.3. Візуальне моделювання архітектури програмного забезпечення 231
12.4. Діаграми UML. Діаграми прецедентів 233
12.5. Моделювання даних і архітектури програмного забезпечення 236
12.6. Діаграми послідовностей та діяльностей 240
12.7. Проектування інтерфейсу користувача 243
12.8. Тестування та оцінювання програмного забезпечення 246
12.9. Системна архітектура. Апаратні та програмні рішення 249

Комп'ютерний словник 251
Алфавітний покажчик 253

Відомості про користування підручником

№ з/п	Прізвище та ім'я учня / учениці	Навчальний рік	Стан підручника	
			на початку року	у кінці року
1				
2				
3				
4				
5				

Навчальне видання
РУДЕНКО Віктор Дмитрович
РЕЧИЧ Наталія Василівна
ПОТІЄНКО Валентина Олександрівна

«ІНФОРМАТИКА (ПРОФІЛЬНИЙ РІВЕНЬ)»
підручник для 11 класу закладів загальної середньої освіти

Рекомендовано Міністерством освіти і науки України

Видано за рахунок державних коштів. Продаж заборонено

Редактор *Л. А. Каюда*. Художнє оформлення *В. І. Труфена*. Технічний редактор *А. В. Пліско*.
Комп'ютерна верстка *І. І. Пікальової*. Коректор *Н. В. Красна*.

Окремі зображення, що використані в оформленні підручника,
розміщені в мережі Інтернет для вільного використання

Підписано до друку 31.05.2019. Формат 84×108/16.
Папір офсетний. Гарнітура Шкільна. Друк офсетний.
Ум. друк. арк. 26,88. Обл.-вид. арк. 35,10.
Тираж 78 584 прим. (1-й запуск 1–30 000). Зам. № 3206-2019/1.

ТОВ Видавництво «Ранок»,
вул. Кібальчича, 27, к. 135, Харків, 61071.
Свідоцтво суб'єкта видавничої справи ДК № 5215 від 22.09.2016.
Адреса редакції: вул. Космічна, 21а, Харків, 61145.
E-mail: office@ranok.com.ua. Тел. (057) 719-48-65, тел./факс (057) 719-58-67.

Підручник надруковано на папері українського виробництва

Надруковано у друкарні ТОВ «ТРІАДА-ПАК»,
пров. Сімферопольський, 6, Харків, 61052 .
Свідоцтво суб'єкта видавничої справи ДК № 5340 від 15.05.2017.
Тел. +38(057) 712-20-00. E-mail: sale@triada.kharkiv.ua

ІНФОРМАТИКА 11

Профільний рівень

Особливості підручника:

- Теоретичний матеріал, адаптований до дворівневої форми подання
- Основні алгоритми створення програм мовою Python
- Вправи практичного спрямування
- Різнорівневі запитання: репродуктивні, дослідницько-пошукові, творчі
- Приклади застосування інформаційних технологій у різних галузях

Інтернет-підтримка дозволить:

- Здійснити онлайн-тестування за кожною темою
- Ознайомитися з додатковими матеріалами до уроків

ВИДАВНИЦТВО
РАНОК

ISBN 978-617-09-5237-0
2314 7706
9 786170 952370

Інтернет-підтримка
interactive.ranok.com.ua