ЗАТВЕРДЖЕНО

Науково-методичною радою
Державного університету
«Житомирська політехніка»
протокол від 19 травня 2026 р.
№ 4

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ
для проведення лабораторних занять з навчальної дисципліни «Обчислювальна техніка та програмування»

для здобувачів вищої освіти освітнього ступеня «бакалавр»
спеціальності G5 «Електроніка, електронні комунікації, приладобудування та радіотехніка»,
освітньо-професійна програма «Інформаційні відеосистеми та системи контролю доступу»

факультет інформаційно-комп’ютерних технологій
кафедра комп’ютерних технологій в медицині те телекомунікаціях

Рекомендовано на засіданні кафедри КТуМтаТ
8 квітня 2026 р., протокол № 3

Розробник: ст. викл. кафедри КТуМтаТ МОРОЗОВ Дмитро
к.т.н., доцент кафедри КТуМтаТ ЧУХОВ Владислав
к.т.н., доцент кафедри КТуМтаТ ГОРШЕНІН Олександр

Житомир
2026

ЗМІСТ

ВСТУП 3
Вимоги до виконання та оформлення звітів з лабораторних робіт 4
Лабораторна робота №1. Основи програмування на мові Python 5
Лабораторна робота №2. Розгалуження та цикли 11
Лабораторна робота №3. Рядки 19
Лабораторна робота №4. Списки 27
Лабораторна робота №5. Функції 36
Лабораторна робота №6. Робота з файлами 50
Лабораторна робота №7. Бібліотеки NumPy та Matplotlib 63
Лабораторна робота №8. Основи об'єктно-орієнтованого програмування в Python 87

ВСТУП

Сучасний етап розвитку науки і техніки характеризується стрімким зростанням обсягів інформації, ускладненням інженерних систем та широким впровадженням цифрових технологій у всі сфери діяльності. За цих умов підготовка фахівців технічних спеціальностей неможлива без ґрунтовного опанування основ обчислювальної техніки та програмування як універсального інструменту розв’язання прикладних і наукових задач.

Формування навичок програмування сприяє розвитку алгоритмічного мислення, здатності до аналізу складних процесів, моделювання та автоматизації обчислень. Вміння створювати ефективні програмні рішення дозволяє майбутнім інженерам підвищувати продуктивність праці, оптимізувати технологічні процеси та впроваджувати інноваційні підходи у професійній діяльності.

Особливе місце в інструментарії майбутнього інженера посідає мова програмування Python, яка завдяки простоті синтаксису, широким можливостям і розвиненій екосистемі бібліотек стала одним із провідних інструментів у галузях інженерії, аналізу даних, штучного інтелекту та наукових досліджень. Її використання у навчальному процесі дозволяє студентам швидко перейти від засвоєння базових концепцій до практичного застосування знань при розв’язанні реальних задач.

Дані методичні вказівки спрямовані на систематизацію теоретичних знань та формування практичних навичок роботи з обчислювальною технікою і мовою Python. Вони покликані забезпечити послідовне оволодіння основами програмування, розвиток самостійності студентів та підготовку їх до ефективного використання сучасних інформаційних технологій у майбутній професійній діяльності.

Вимоги до виконання та оформлення звітів з лабораторних робіт

Підготовка до кожної роботи проводиться в позааудиторний час.

Студенти опрацьовують лекційний матеріал, знайомляться із загальними відомостями, пишуть необхідні програми відповідно до отриманого варіанта індивідуального завдання.

Усі роботи виконуються на мові програмування Python. Під час занять студенти проводять тестування написаних програм, тобто запускають їх в інтегрованому середовищі розробки на персональних комп'ютерах, займаються налагодженням і виконують необхідні розрахунки.

Якщо студент виконує завдання лабораторних робіт на власному комп'ютері, він має самостійно встановити інтерпретатор мови Python (www.python.org/downloads) та редактор або інтегроване середовище розробки. В якості середовища розробки студенту пропонується використовувати PyCharm (www.jetbrains.com/pycharm) або редактор Visual Studio Code (code.visualstudio.com) чи будь які інші редактори коду для роботи з мовою Python з відповідними програмними модулями і налаштуваннями.

Після виконання роботи студент оформляє звіт, який складається з таких частин:

1. Титульний аркуш
2. Тема і мета роботи
2. Завдання на лабораторну роботу
3. Текст програми
4. Результати виконання програми (скріншоти терміналу, графіки тощо)
5. Висновки.

Приклад оформлення звіту про виконання лабораторної роботи наведено на сторінці дисципліни на освітньому порталі.

Титульний аркуш має містити інформацію про назву дисципліни, номер і тему лабораторної роботи, прізвище, ім'я і по-батькові виконавця, шифр і номер його групи і шифр роботи в рамці основного напису.

В шифрі роботи потрібно зазначити:

1. Шифр групи (виділено синьою рамкою).
2. Номер варіанту індивідуального завдання, якщо воно передбачено в роботі (виділено зеленою рамкою).
3. Три останні цифри залікової книжки (виділено червоною рамкою)

Робота оформлюється в електронному вигляді і надсилається в форматі pdf разом з файлами з кодом програм відповідних завдань лабораторної роботи на адресу електронної пошти викладача для перевірки.

Для кожного завдання пишеться окрема програма на Python. Імена файлів з кодом мають відповідати номеру завдання: *task1.1.py, task1.2.py* тощо.

Під час захисту роботи необхідно відповісти на контрольні питання і вміти пояснити роботу програми.

Лабораторна робота №1.

Основи програмування на мові Python

Мета роботи: ознайомитися з основними концепціями програмування на мові Python. Навчитися оголошувати змінні різних типів, виконувати базові математичні операції та використовувати вбудований модуль math.

Теоретичні відомості:

Python — це сучасна мова програмування високого рівня із простим, читабельним синтаксисом. Вона широко використовується у веб-розробці, аналізі даних, штучному інтелекті, автоматизації та навчанні.
Основні переваги Python:

- Простий синтаксис спрощує написання і читання коду
- Великий набір вбудованих функцій та бібліотек
- Підтримка різних парадигм програмування
- Крос-платформенність (Windows, macOS, Linux)

Функція `print()` виводить інформацію на екран (консоль). Синтаксис:

```
# Виведення рядка тексту
print('Привіт, світ!')

# Виведення кількох значень через кому (роздільник — пробіл)
print('Сума:', 3 + 5)

# Зміна роздільника між значеннями
print('a', 'b', 'c', sep='-')    # Виведе: a-b-c

# Зміна символу кінця рядка
print('Рядок 1', end=' ')
print('Рядок 2')                 # Виведе: Рядок 1 Рядок 2
```

Після запуску програми вивід в консолі покаже результат виконання коду:

```
Привіт, світ!
Сума: 8
a-b-c
Рядок 1 Рядок 2
```

Змінна — це іменована область пам'яті для зберігання даних. У Python змінні не потребують оголошення типу — він визначається автоматично під час присвоєння значення.

Правила іменування змінних:

- Може містити літери, цифри та знак підкреслення _
- Не може починатися з цифри
- Регістр має значення: `age` та `Age` — різні змінні
- Не можна використовувати зарезервовані слова (`if`, `for`, `while` тощо)

```python
# Присвоєння значень змінним
name = 'Олена'
age = 20
height = 1.68
is_student = True

# Множинне присвоєння
x = y = z = 0

# Паралельне присвоєння
a, b, c = 1, 2, 3

# Виведення змінних
print('Ім\'я:', name)
print('Вік:', age)
```

```
Ім'я: Олена
Вік: 20
```

Python має кілька вбудованих типів даних. Розглянемо основні з них:

`int` — цілі числа (без дробової частини):

```python
population = 1000000
temperature = -5
print(type(population))    # <class 'int'>
```

`float` — числа з плаваючою крапкою (дробові числа):

```python
price = 29.99
pi = 3.14159
print(type(price))         # <class 'float'>
```

`str` — рядки (текст у лапках):

```python
greeting = 'Привіт!'
city = "Київ"
multiline = '''Перший рядок
Другий рядок'''
print(type(greeting))      # <class 'str'>
```

`bool` — логічний тип (`True` або `False`):

```python
is_active = True
is_empty = False
print(type(is_active))     # <class 'bool'>
```

Функція `type()` дозволяє перевірити тип будь-якої змінної або значення:

```python
x = 42
print(type(x))             # <class 'int'>
print(type(3.14))          # <class 'float'>
print(type('hello'))       # <class 'str'>
print(type(True))          # <class 'bool'>
```

Іноді потрібно змінити тип даних. Для цього використовуються вбудовані функції перетворення:

```
# Перетворення у ціле число
x = int('42')              # рядок -> int
y = int(3.9)               # float -> int (відкидає дробову частину!)
print(x, y)                # 42  3

# Перетворення у дробове число
a = float('3.14')          # рядок -> float
b = float(5)               # int -> float
print(a, b)                # 3.14  5.0
# Перетворення у рядок
s = str(100)               # int -> str
t = str(3.14)              # float -> str
print(s + ' грн')          # 100 грн
```

Функція `input()` зупиняє виконання програми та очікує введення користувача з клавіатури. Вона завжди повертає значення типу `str` (рядок).

```
# Базове використання
name = input('Введіть ваше ім\'я: ')
print('Привіт,', name)

# ВАЖЛИВО: input() повертає рядок!
age_str = input('Введіть вік: ')
age = int(age_str)         # Обов'язково конвертуємо у int

# Або одразу:
age = int(input('Введіть вік: '))

# Введення дробового числа
height = float(input('Введіть зріст (м): '))
```

Python підтримує стандартні арифметичні операції:

```
a = 15
b = 4

print(a + b)               # 19   — додавання
print(a - b)               # 11   — віднімання
print(a * b)               # 60   — множення
print(a / b)               # 3.75 — ділення (завжди float)
print(a // b)              # 3    — ціле ділення (без залишку)
print(a % b)               # 3    — остача від ділення
print(a ** b)              # 50625 — піднесення до степеня

# Скорочені оператори присвоєння
x = 10
x += 5     # x = x + 5  -> 15
x -= 3     # x = x - 3  -> 12
x *= 2     # x = x * 2  -> 24
x //= 7    # x = x // 7 -> 3
```

Форматування рядків (f-рядки) часто використовується для виведення інформації в термінал в потрібному форматі. Сучасний та зручний спосіб вставляти змінні в рядки — це f-рядки (починаються з літери `f` перед лапками):

```
name = 'Іван'
age = 21
gpa = 4.75

# Проста підстановка змінних
print(f'Привіт, {name}! Тобі {age} років.')

# Форматування чисел
print(f'Середній бал: {gpa:.2f}')   # 4.75

# Вирази всередині {}
a, b = 7, 3
print(f'{a} + {b} = {a + b}')       # 7 + 3 = 10
print(f'Площа: {a * b} кв.м')
```

Модуль `math` надає доступ до математичних функцій та констант. Перед використанням його потрібно імпортувати командою `import math`.

```
import math

# Константи
print(math.pi)              # 3.141592653589793
print(math.e)               # 2.718281828459045
# Функції заокруглення
print(math.floor(4.7))      # 4  — округлення вниз
print(math.ceil(4.2))       # 5  — округлення вгору
print(round(4.567, 2))      # 4.57 — вбудоване round()

# Корінь та степінь
print(math.sqrt(49))        # 7.0 — квадратний корінь
print(math.pow(2, 10))      # 1024.0 — степінь

# Тригонометрія (аргументи в радіанах)
angle_deg = 45
angle_rad = math.radians(angle_deg)
print(math.sin(angle_rad)) # 0.7071...
print(math.cos(angle_rad)) # 0.7071...

# Логарифми
print(math.log(100, 10))    # 2.0  — log base 10
print(math.log(math.e))     # 1.0  — натуральний логарифм

# Модуль (абсолютне значення)
print(abs(-15))             # 15   — вбудована функція
print(math.fabs(-3.7))      # 3.7  — math версія
```

Коментарі допомагають пояснити логіку програми. Python ігнорує їх під час виконання:

```
# Це однорядковий коментар
x = 5  # Коментар після коду
'''
Це багаторядковий коментар.
Зазвичай використовується для
документування функцій та класів.
'''
```

Завдання на лабораторну роботу

1.1. Напишіть програму, яка виводить на екран:
- ваше прізвище, ім'я та по батькові;
- назву вашої спеціальності та номер групи;
- поточний рік навчання.

Кожен рядок виводьте за допомогою окремого виклику `print()`. Також виведіть усі три значення в одному рядку через функцію `print()` використавши параметр `sep`.

1.2. Оголосіть по дві змінні кожного з типів: `int`, `float`, `str`, `bool`. Виведіть значення та тип кожної змінної за допомогою функцій `print()` та `type()`. Наприклад:

```
x = 42
print(f'Значення: {x}, Тип: {type(x)}')
```

1.3. Напишіть програму, яка:
- запитує у користувача ім'я (рядок);
- запитує рік народження (ціле число);
- обчислює вік (поточний рік мінус рік народження);
- виводить привітання у форматі: `"Привіт, [ім'я]! Тобі [вік] років."`

1.4. Напишіть програму-калькулятор, яка:
- зчитує два дробових числа з клавіатури (float);
- виводить результати всіх 7 арифметичних операцій: +, -, *, /, //, %, **;
- форматує виведення так, щоб результати дробових операцій показувались з 2 знаками після коми (використовуйте f-рядки).

1.5.Напишіть програму для обчислення характеристик прямокутника:
- зчитайте довжину та ширину прямокутника (float);
- обчисліть і виведіть периметр, площу та довжину діагоналі (використайте функцію `math.sqrt()`).

Результати виведіть у зрозумілому форматі із поясненнями

1.6. Напишіть програму для перетворення температури:
- зчитайте температуру в градусах Цельсія;
- перетворіть у градуси Фаренгейта за формулою: F = C * 9/5 + 32;
- перетворіть у Кельвіни: K = C + 273.15.

Виведіть усі три значення, заокруглені до 2 знаків після коми.

1.7. Напишіть програму для розв'язання наступної задачі:
Користувач вводить координати двох точок на площині (x1, y1) та (x2, y2). Програма повинна:

- обчислити відстань між точками за формулою Евкліда:
  `d = sqrt((x2-x1)^2 + (y2-y1)^2)`
- обчислити координати середини відрізка:
  `(xm, ym) = ((x1+x2)/2, (y1+y2)/2)`
- обчислити периметр і площу трикутника, утвореного двома введеними точками та початком координат (0, 0) (використайте формулу Герона)

Вивести усі результати із поясненнями та форматуванням до 4 знаків після коми

Підказка: формула Герона `S = sqrt(s*(s-a)*(s-b)*(s-c))`,
де `s = (a+b+c)/2`,
`a`, `b`, `c` *— сторони трикутника.*

Контрольні запитання

1. Що таке змінна у програмуванні? Які правила іменування змінних існують у Python?
2. Назвіть основні вбудовані типи даних у Python. Чим відрізняється `int` від `float`?
3. Що повертає функція `input()`? Чому важливо перетворювати результат `input()` перед виконанням арифметичних операцій?
4. Яка різниця між оператором ділення `/` та цілочисельного ділення `//`? Наведіть приклад.
5. Що виводить вираз `print(type(3.14))`? Поясніть результат.
6. Що таке f-рядок? Як вставити значення змінної у текстовий рядок за допомогою f-рядка?
7. Як підключити модуль `math`? Наведіть приклади двох функцій та двох констант цього модуля.
8. Яка різниця між `math.floor()`, `math.ceil()` та `round()`? Наведіть приклади.
9. Що таке коментар у коді? Як записати однорядковий коментар у Python?
10. Який параметр визначає знак роздільника в функції `print()`?

Лабораторна робота №2.
Розгалуження та цикли

Мета роботи: вивчити конструкції розгалуження (`if-else`, `if-elif-else`, тернарний оператор, `match-case`) та навчитися їх застосовувати для керування потоком виконання програми залежно від умов; ознайомитися з циклами `while` та `for`, зрозуміти принципи їхньої роботи; навчитися використовувати оператори `break` і `continue` для керування ітерацією.

Теоретичні відомості:

Логічні вирази та оператори порівняння є невід'ємною частиною будь якої високорівневої мови програмування. Тому перед вивченням розгалужень необхідно розглянути логічні вирази — вирази, що повертають True або False. Вони є основою будь-якої умови.

Оператори порівняння:

```
a, b = 10, 20

print(a == b)    # False  — рівність
print(a != b)    # True   — нерівність
print(a < b)     # True   — менше
print(a > b)     # False  — більше
print(a <= 10)   # True   — менше або рівне
print(a >= 10)   # True   — більше або рівне
```

Логічні оператори для об'єднання умов:

```
x = 15

# and — True, якщо ОБИДВІ умови True
print(x > 10 and x < 20)   # True
print(x > 10 and x > 20)   # False

# or — True, якщо ХОЧА Б ОДНА умова True
print(x < 5 or x > 10)     # True
print(x < 5 or x < 0)      # False

# not — інвертує логічне значення
print(not (x > 10))        # False
print(not False)            # True
```

Оператор `if` дозволяє виконати блок коду тільки тоді, коли умова є істинною (`True`). Блок `else` виконується у коли умова є хибною (`False`).

Важливо: у Python використовуються відступи (4 пробіли або Tab) визначають, які рядки належать до блоку (рівні вкладеності). Це не просто стиль — це синтаксична вимога!

```
# Простий if
temperature = 35
if temperature > 30:
    print('Спекотно!')
```

```
# if-else
age = int(input('Введіть вік: '))
if age >= 18:
    print('Доступ дозволено')
else:
    print('Доступ заборонено — вам ще немає 18')

# Перевірка парності числа
n = int(input('Введіть число: '))
if n % 2 == 0:
    print(f'{n} — парне')
else:
    print(f'{n} — непарне')
```

Якщо варіантів більше двох, використовують `elif` (скорочення від `else if`). Python перевіряє умови зверху вниз і виконує перший блок, умова якого є `True`. Решта блоків пропускаються.

```
score = int(input('Введіть оцінку (0-100): '))

if score >= 90:
    grade = 'Відмінно'
elif score >= 75:
    grade = 'Добре'
elif score >= 60:
    grade = 'Задовільно'
else:
    grade = 'Незадовільно'

print(f'Оцінка: {grade}')

# Визначення пори року
month = int(input('Введіть місяць (1-12): '))

if month in [12, 1, 2]:
    season = 'Зима'
elif month in [3, 4, 5]:
    season = 'Весна'
elif month in [6, 7, 8]:
    season = 'Літо'
else:
    season = 'Осінь'

print(f'Місяць {month}: {season}')
```

Блоки `if` можна вкладати один в одного. Однак глибоке вкладення ускладнює читання коду (*намагайтеся уникати більше 2–3 рівнів вкладеності!*).

```
# Перевірка трикутника та його типу
a = float(input('Сторона a: '))
b = float(input('Сторона b: '))
c = float(input('Сторона c: '))

# Умова існування трикутника
if a + b > c and a + c > b and b + c > a:
    print('Трикутник існує')
    if a == b == c:
        print('Тип: рівносторонній')
    elif a == b or b == c or a == c:
        print('Тип: рівнобедрений')
    else:
        print('Тип: різносторонній')
else:
    print('Такий трикутник не існує')
```

Тернарний оператор — це скорочений запис `if-else` в одному рядку. Зручний для простих умов, коли потрібно вибрати одне з двох значень.

```
# Повна форма if-else
x = 7
if x % 2 == 0:
    result = 'парне'
else:
    result = 'непарне'

# Еквівалентний тернарний запис (одним рядком)
result = 'парне' if x % 2 == 0 else 'непарне'
print(result)              # непарне

# Знаходження максимуму двох чисел
a, b = 15, 28
maximum = a if a > b else b
print(f'Максимум: {maximum}')  # 28

# Абсолютне значення без abs()
n = -9
abs_n = n if n >= 0 else -n
print(abs_n)
```

Оператор `match-case` з'явився у Python 3.10. Він схожий на `switch-case` з інших мов програмування і дозволяє порівнювати значення з кількома шаблонами. Оператор `_` у `case` є шаблоном за замовчуванням (аналог `else`).

```
# Перевірка дня тижня
day = int(input('Введіть номер дня (1-7): '))

match day:
    case 1:
```

```
        print('Понеділок — початок тижня')
    case 2:
        print('Вівторок')
    case 3:
        print('Середа')
    case 4:
        print('Четвер')
    case 5:
        print('П\'ятниця — кінець робочого тижня!')
    case 6 | 7:              # Кілька значень через |
        print('Вихідний день')
    case _:                  # За замовчуванням
        print('Помилка: введіть число від 1 до 7')

# match з рядками
command = input('Команда (start/stop/pause): ')

match command:
    case 'start':
        print('Запуск програми...')
    case 'stop':
        print('Зупинка програми...')
    case 'pause':
        print('Пауза...')
    case _:
        print(f'Невідома команда: {command}')
```

Цикл `while` виконує блок коду доти, доки умова залишається `True`. Якщо умова хибна від початку — тіло циклу не виконається жодного разу. *Обов'язково стежте, щоб умова колись стала False, інакше програма зациклиться.*

```
# Лічильник від 1 до 5
i = 1
while i <= 5:
    print(i, end=' ')       # 1 2 3 4 5
    i += 1                  # ВАЖЛИВО: змінювати лічильник!

# Сума цифр числа
n = int(input('Введіть число: '))
total = 0
while n > 0:
    total += n % 10         # Додаємо останню цифру
    n //= 10                # Відкидаємо останню цифру
print(f'Сума цифр: {total}')

# Валідація введення (цикл поки не введуть правильне значення)
while True:
    age = int(input('Введіть вік (1-120): '))
    if 1 <= age <= 120:
        break               # Вихід з циклу при правильному введенні
    print('Некоректний вік, спробуйте знову')
print(f'Ваш вік: {age}')
```

Цикл `for` використовується для перебору елементів послідовності (рядка, списку, діапазону тощо). Функція `range()` генерує послідовність чисел і є найчастішим аргументом `for`. Найчастіше можна зустріти в одному з трьох варіанти виклику:

```
range(stop)
range(start, stop)
range(start, stop, step)
```

```python
# range(stop) — від 0 до stop-1
for i in range(5):
    print(i, end=' ')      # 0 1 2 3 4

# range(start, stop) — від start до stop-1
for i in range(1, 6):
    print(i, end=' ')      # 1 2 3 4 5

# range(start, stop, step) — з кроком
for i in range(0, 20, 5):
    print(i, end=' ')      # 0 5 10 15

# Зворотній відлік (від'ємний крок)
for i in range(5, 0, -1):
    print(i, end=' ')      # 5 4 3 2 1

# Перебір рядка (рядок — це послідовність символів)
word = 'Python'
for char in word:
    print(char, end='-')   # P-y-t-h-o-n-

# Таблиця множення (вкладені цикли)
for i in range(1, 4):
    for j in range(1, 4):
        print(f'{i}x{j}={i*j}', end='  ')
    print()                # Переносимо рядок після кожного і
```

Оператори `break` і `continue` змінюють звичайний хід виконання циклу і можуть використовуватися як у `while`, так і у `for`.

Оператор `break` повністю зупиняє цикл і передає керування на перший рядок після нього, незалежно від того, чи виконана умова циклу.

```python
# Пошук першого від'ємного числа в послідовності
numbers = [5, 12, 3, -7, 8, -2, 1]

for num in numbers:
    if num < 0:
        print(f'Знайдено від\'ємне число: {num}')
        break              # Зупиняємось після першого збігу
    print(f'{num} — позитивне')

# Перевірка простоти числа
n = int(input('Число: '))
is_prime = True
```

```python
for i in range(2, int(n**0.5) + 1):
    if n % i == 0:
        is_prime = False
        break              # Знайшли дільник — далі перевіряти не потрібно
print('Просте' if is_prime and n > 1 else 'Не просте')
```

Оператор `continue` пропускає решту поточної ітерації та переходить до наступної перевірки умови (`while`) або наступного елемента (`for`).

```python
# Виведення тільки парних чисел від 1 до 10
for i in range(1, 11):
    if i % 2 != 0:
        continue           # Пропускаємо непарні
    print(i, end=' ')      # 2 4 6 8 10

# Підрахунок позитивних чисел (ігноруємо нулі та від'ємні)
data = [3, -1, 0, 7, -5, 2, 0, 9]
count = 0
total = 0
for x in data:
    if x <= 0:
        continue           # Пропускаємо нульові та від'ємні
    count += 1
    total += x

print(f'Позитивних: {count}, Сума: {total}')
```

У Python є унікальна особливість: цикли `while` та `for` можуть мати блок `else`. Він виконується, якщо цикл завершився природно (без `break`). Якщо виконувався `break` — блок `else` пропускається.

```python
# Пошук елемента у списку
numbers = [4, 7, 2, 9, 1]
target = int(input('Шукаємо число: '))

for num in numbers:
    if num == target:
        print(f'Знайдено: {target}')
        break
else:
    # Виконається ТІЛЬКИ якщо break не спрацював
    print(f'{target} у списку відсутнє')

# while ... else: перевірка пароля
correct = 'qwerty123'
attempts = 3

while attempts > 0:
    pwd = input('Пароль: ')
    if pwd == correct:
        print('Вхід успішний!')
        break
```

```
    attempts -= 1
    print(f'Невірно. Залишилось спроб: {attempts}')
else:
    print('Акаунт заблоковано')
```

Завдання на лабораторну роботу

2.1. Класифікація числа. Напишіть програму, яка зчитує ціле число і виводить:
- чи є воно додатним, від'ємним або нулем?
- чи є воно парним або непарним (для нуля — лише "нуль")?
- Чи ділиться воно одночасно на 3 і на 5

Використайте `if-elif-else` та тернарний оператор (хоча б один раз).

2.2. Калькулятор з вибором операції. Напишіть програму-калькулятор:
- зчитайте два числа та символ операції (+, -, *, /, //, %, **);
- використайте `match-case` для вибору операції;
- передбачте захист від ділення на нуль для операторів "`/`" і "`//`": перевіряйте, чи `b != 0`);
- для невідомого символу виведіть повідомлення про помилку.

2.3. Підрахунок суми з циклом `while`. Користувач вводить числа одне за одним. Програма підсумовує їх. Введення числа 0 зупиняє цикл. Після завершення виведіть суму, кількість введених чисел (не рахуючи 0) та середнє арифметичне.

2.4. Таблиця множення з форматуванням. За допомогою вкладених циклів `for` виведіть таблицю множення від 1 до N, де N вводить користувач (від 2 до 9). Вирівняйте стовпці так, щоб таблиця виглядала охайно (використайте f-рядки з вирівнюванням, наприклад `f"{i*j:4d}"`). Пропустіть виведення, якщо `i > j` (виводьте тільки верхню трикутну частину таблиці, використовуючи `continue`).

2.5. Визначення простих чисел. Напишіть програму, яка:
- зчитує ціле число N (> 1);
- виводить усі прості числа від 2 до N;

Для перевірки простоти використайте цикл for з break та else
Підрахуйте та виведіть загальну кількість знайдених простих чисел

2.6. Напишіть гру "Вгадай число":
- задайте "секретне" число (наприклад, 42) у коді програми;
- цикл `while` запитує у користувача число від 1 до 100;
- після кожної спроби виводьте підказку: "більше" або "менше";
- якщо вгадав — вивести привітання та кількість спроб, вийти з циклу (`break`).

Обмежте кількість спроб до 7. Якщо вичерпано — виведіть секретне число (`else` у `while`).

2.7. Аналіз числової послідовності. Напишіть програму, яка зчитує числа від користувача (по одному, завершення — введення слова "стоп") та виконує комплексний аналіз:
- підраховує кількість: додатних, від'ємних, нульових;
- знаходить мінімальне і максимальне значення серед введених;

- обчислює суму та середнє арифметичне тільки для додатних чисел;
- виводить скільки чисел кратні 3, скільки — кратні 5, скільки — одночасно кратні 3 і 5;
- виводить зведену таблицю результатів у форматованому вигляді.

Вимоги до реалізації: використайте `while True` з `break` для завершення, `continue` для пропуску нечислових введень (спроба конвертації у `float` через `try-except` або перевірку через умову), `if-elif-else` для класифікації, f-рядки для виведення.

Контрольні запитання

1. Що таке розгалуження? Яку роль відіграють відступи у конструкціях `if-elif-else` у Python?
2. Чим відрізняється конструкція `if-elif-else` від кількох окремих `if`? Коли доцільно використовувати кожен варіант?
3. Що таке тернарний оператор? Запишіть приклад тернарного виразу для знаходження максимуму двох чисел a і b.
4. У яких версіях Python з'явився оператор `match-case`? Що означає шаблон `_` у блоці `case`?
5. Чим відрізняється цикл `while` від циклу `for`? Наведіть приклади задач, де кожен з них підходить краще.
6. Що станеться, якщо у циклі `while` не змінювати значення змінної, від якої залежить умова? Як називається такий стан?
7. Поясніть роботу оператора `break`. Як він взаємодіє з конструкцією `for...else`?
8. Поясніть роботу оператора `continue`. Чим він відрізняється від `break`?
9. Що виведе наступний код:
   ```
   for i in range(2, 10, 3):
       print(i, end=' ')
   ```
   Поясніть роботу `range()`.
10. Коли виконується блок `else` у циклі? Наведіть практичний приклад його використання.

Лабораторна робота №3.

Рядки

Мета роботи: вивчити рядковий тип даних `str` у Python: способи створення рядків, індексування, зрізи та базові операції над рядками (конкатенація, повторення, перевірка входження); ознайомитися з основними рядковими методами Python та навчитися їх практично застосовувати; сформувати навички обробки текстових даних.

Теоретичні відомості:

Рядок (`str`) — це незмінна послідовність символів Unicode. У Python рядки можна записувати в одинарних, подвійних або потрійних лапках. Незмінність означає, що жоден метод не змінює оригінальний рядок — всі вони повертають новий.

```
# Одинарні та подвійні лапки — рівноцінні
s1 = 'Привіт'
s2 = "Привіт"

# Потрійні лапки — багаторядковий рядок
s3 = '''Перший рядок
Другий рядок
Третій рядок'''

# Спеціальні символи (escape-послідовності)
s4 = 'Рядок 1\nРядок 2'      # \n — новий рядок
s5 = 'Колонка1\tКолонка2'    # \t — табуляція
s6 = 'Вона сказала: \'OK\''  # \' — екранована лапка
s7 = 'Шлях: C:\\Users'       # \\ — зворотній слеш

# «Сирий» рядок (raw) — зворотні слеші не є escape
path = r'C:\Users\Admin\file.txt'
print(path)   # C:\Users\Admin\file.txt

# Порожній рядок
empty = ''
print(len(empty))    # 0
```

Кожен символ рядка має індекс — позицію з 0. Python також підтримує від'ємні індекси: -1 — останній символ, -2 — передостанній і т.д.

```
word = 'Python'
#         P  y  t  h  o  n
# індекс  0  1  2  3  4  5
# від'єм -6 -5 -4 -3 -2 -1

print(word[0])      # P   — перший символ
print(word[3])      # h   — четвертий символ
print(word[-1])     # n   — останній символ
print(word[-2])     # o   — передостанній
```

Зріз (`slice`) дозволяє отримати підрядок. Синтаксис: `рядок[start : stop : step]`

```
s = 'Hello, World!'

print(s[0:5])       # Hello   — символи 0..4
print(s[7:])        # World!  — від 7 до кінця
print(s[:5])        # Hello   — від початку до 4
print(s[-6:])       # World!  — останні 6 символів
print(s[::2])       # Hlo ol! — кожен другий
print(s[::-1])      # !dlroW ,olleH — реверс рядка
print(s[7:12])      # World
```

Базові операції з рядками

```
a = 'Hello'
b = ' World'
# Конкатенація (склеювання) — оператор +
c = a + b
print(c)            # Hello World

# Повторення — оператор *
print('-' * 20)     # --------------------
print('ab' * 3)     # ababab

# Довжина рядка
print(len(c))       # 11

# Перевірка входження — оператори in та not in
print('ello' in a)        # True
print('xyz' in a)         # False
print('xyz' not in a)     # True

# Порівняння рядків (лексикографічне, за Unicode)
print('abc' == 'abc')     # True
print('abc' < 'abd')      # True  (c < d)
print('A' < 'a')          # True  (65 < 97 у Unicode)

# Перетворення числа у рядок і навпаки
n = 42
s = str(n) + ' — відповідь'
print(s)            # 42 — відповідь
```

Python пропонує кілька способів вставити змінні у рядок. Сучасний і рекомендований — f-рядки

```
name = 'Олена'
score = 94.333
# f-рядок (найсучасніший спосіб)
print(f'Студент: {name}, бал: {score:.1f}')
# -> Студент: Олена, бал: 94.3
# Вирівнювання у f-рядках
for item, price in [('Молоко', 38.5), ('Хліб', 22), ('Масло', 115)]:
    print(f'{item:<10} {price:>7.2f} грн')
# Молоко        38.50 грн
```

```
# Хліб          22.00 грн
# Масло        115.00 грн

# Метод .format()
print('Привіт, {}! Тобі {} років.'.format(name, 21))

# % — старий стиль (зустрічається у старому коді)
print('Бал: %.2f' % score)    # Бал: 94.33
```

Методи зміни регістру та перевірки
Усі наведені нижче методи повертають новий рядок і не змінюють оригінал.

Метод / синтаксис	Опис та приклад
`s.upper()`	Всі символи у верхній регістр: `'hello'.upper()` → `'HELLO'`
`s.lower()`	Всі символи у нижній регістр: `'WORLD'.lower()` → `'world'`
`s.title()`	Кожне слово з великої літери: `'py code'.title()` → `'Py Code'`
`s.capitalize()`	Перша літера велика, решта — малі: `'pYTHON'.capitalize()` → `'Python'`
`s.swapcase()`	Міняє регістр кожної літери: `'PyThOn'.swapcase()` → `'pYtHoN'`
`s.isupper()`	True, якщо всі літери — великі: `'ABC'.isupper()` → `True`
`s.islower()`	True, якщо всі літери — малі: `'abc'.islower()` → `True`
`s.istitle()`	True, якщо кожне слово з великої: `'Hello World'.istitle()` → `True`

```
text = 'привіт, СВІТ!'
print(text.upper())       # ПРИВІТ, СВІТ!
print(text.lower())       # привіт, світ!
print(text.title())       # Привіт, Світ!
print(text.capitalize())  # Привіт, світ!

# Практичне застосування: порівняння без урахування регістру
user_input = input('Місто: ')
if user_input.lower() == 'київ':
    print('Столиця України!')
```

Методи пошуку та перевірки вмісту

Метод / синтаксис	Опис та приклад
`s.find(sub)`	Індекс першого входження sub, або -1: `'python'.find('th')` → `2`
`s.rfind(sub)`	Індекс останнього входження sub: `'abab'.rfind('ab')` → `2`
`s.index(sub)`	Працює як `find()`, але виводить `ValueError` якщо не знайдено
`s.count(sub)`	Кількість входжень sub: `'banana'.count('a')` → `3`
`s.startswith(p)`	`'Hello'.startswith('He')` → `True`
`s.endswith(p)`	`'file.txt'.endswith('.txt')` → `True`
`s.isdigit()`	True, якщо всі символи — цифри: `'123'.isdigit()` → `True`
`s.isalpha()`	True, якщо всі символи — літери: `'abc'.isalpha()` → `True`
`s.isalnum()`	True, якщо лише літери і цифри: `'abc123'.isalnum()` → `True`
`s.isspace()`	True, якщо рядок містить лише пробіли/\n/\t

```python
email = 'user@example.com'

# Пошук символу @ в email
pos = email.find('@')
print(pos)                      # 4
print(email[:pos])              # user  (логін)
print(email[pos+1:])            # example.com  (домен)

# Підрахунок голосних у слові
word = 'програмування'
vowels = 'аеєиіїоуюя'
count = sum(1 for ch in word if ch.lower() in vowels)
print(f'Голосних: {count}')

# Перевірка, чи рядок — ціле число
s = input('Введіть число: ')
if s.lstrip('-').isdigit():
    print(f'Ціле число: {int(s)}')
else:
    print('Це не ціле число')
```

Методи заміни та очищення

Метод / синтаксис	Опис та приклад
`s.replace(old, new)`	Замінює old на new: `'hello'.replace('l','r')` → `'herro'`
`s.replace(old, new, n)`	Замінює не більше n входжень
`s.strip()`	Видаляє пробіли/\n/\t з обох кінців
`s.lstrip()`	Видаляє з лівого краю
`s.rstrip()`	Видаляє з правого краю
`s.strip(chars)`	`'***hi***'.strip('*')` → `'hi'`
`s.removeprefix(p)`	Видаляє prefix якщо є
`s.removesuffix(s)`	Видаляє suffix якщо є

```python
# Очищення введених даних
raw = '  Іван Петренко  '
name = raw.strip()
print(f'|{name}|')              # |Іван Петренко|

# Заміна символів
text = 'кіт сів на килим'
print(text.replace('кіт', 'пес'))  # пес сів на килим

# Заміна тільки першого входження
s = 'a-b-c-d'
print(s.replace('-', ':', 1))  # a:b-c-d

# Видалення конкретних символів
dirty = '###важлива інформація###'
clean = dirty.strip('#')
```

```
print(clean)                  # важлива інформація

# removeprefix / removesuffix
filename = 'report_2024.csv'
print(filename.removesuffix('.csv'))   # report_2024
```

Розбиття та об'єднання рядків

Методи `split()` та `join()` — одні з найуживаніших у практичному програмуванні. `split()` перетворює рядок на список підрядків, `join()` — навпаки.

Метод / синтаксис	Опис та приклад
s.split(sep)	Розбиває за роздільником `sep` (за замовч. — пробіл)
s.split(sep, n)	Розбиває не більше `n` разів
s.rsplit(sep, n)	Розбиває справа від `sep`
s.splitlines()	Розбиває за символами нового рядка `\n`
sep.join(list)	Об'єднує елементи списку рядком `sep`

```
# split() — рядок у список
sentence = 'Python дуже зручна мова'
words = sentence.split()         # split за пробілом
print(words)    # ['Python', 'дуже', 'зручна', 'мова']
print(len(words))  # 4  — кількість слів

# split() за довільним роздільником
csv_line = 'Іван,22,Київ,відмінник'
parts = csv_line.split(',')
print(parts[0])  # Іван
print(parts[2])  # Київ

# join() — список у рядок
fruits = ['яблуко', 'груша', 'слива']
print(', '.join(fruits))   # яблуко, груша, слива
print(' | '.join(fruits))  # яблуко | груша | слива
print(''.join(fruits))     # яблукогрушаслива

# Практика: реверс слів у реченні
text = 'один два три'
reversed_text = ' '.join(text.split()[::-1])
print(reversed_text)       # три два один
```

Вирівнювання та заповнення

Метод / синтаксис	Опис та приклад
`s.center(w)`	Центрує рядок у полі ширини `w`: `'hi'.center(10)` → `'    hi    '`
`s.center(w, ch)`	`'hi'.center(10, '-')` → `'----hi----'`
`s.ljust(w)`	Вирівнювання ліворуч з пробілами до ширини `w`
`s.rjust(w)`	Вирівнювання праворуч
`s.zfill(w)`	Доповнює нулями зліва: `'42'.zfill(6)` → `'000042'`

```python
# Красиве виведення таблиці
header = ' ЗВІТ УСПІШНОСТІ '
print(header.center(30, '='))
# ======= ЗВІТ УСПІШНОСТІ =======

subjects = [('Математика', 95), ('Фізика', 87), ('Хімія', 72)]
for subj, mark in subjects:
    print(f'{subj:<14} {mark:>3}')
# Математика      95
# Фізика          87
# Хімія           72

# zfill для форматування номерів
for i in range(1, 6):
    print(f'Файл №{str(i).zfill(3)}.txt')
# Файл №001.txt  Файл №002.txt ...
```

Рядок є ітерованою послідовністю, тому його можна перебирати циклом `for`. Функції `ord()` та `chr()` дозволяють працювати з Unicode-кодами символів.

```python
# Перебір символів рядка
word = 'code'
for char in word:
    print(char, end=' ')    # c o d e

# Перебір з індексом через enumerate()
for i, char in enumerate('Python'):
    print(f'{i}: {char}')
# 0: P  1: y  2: t  3: h  4: o  5: n

# ord() — Unicode-код символу
print(ord('A'))    # 65
print(ord('a'))    # 97
print(ord('П'))    # 1055

# chr() — символ за Unicode-кодом
print(chr(65))     # A
print(chr(1055))   # П

# Шифр Цезаря (зсув літер на 3)
text = 'hello'
encrypted = ''.join(chr(ord(c) + 3) for c in text)
print(encrypted)   # khoor

# Підрахунок символів вручну (без count)
sentence = 'programming is fun'
letter_count = {}
for ch in sentence:
    if ch != ' ':
        letter_count[ch] = letter_count.get(ch, 0) + 1
print(letter_count)
```

Завдання на лабораторну роботу

3.1. Аналіз рядка. Зчитайте будь-який рядок із клавіатури та виведіть:
- довжину рядка (кількість символів);
- рядок у верхньому та нижньому регістрі;
- перший та останній символи рядка (через індексування);
- чи починається і чи закінчується рядок на задану літеру (запитайте літеру у користувача);
- кількість входжень заданого символу у рядку.

3.2. Обробка імені.Зчитайте повне ім'я людини у форматі "Прізвище Ім'я По батькові" (один рядок). Програма повинна:
- очистити рядок від зайвих пробілів на початку і в кінці (`.strip()`);
- розбити на складові за допомогою `.split()` та вивести кожну окремо;
- сформувати скорочену форму: `"Прізвище І.П."` (`Прізвище + перша літера імені + '.' + перша літера по батькові + '.'`);
- вивести все прізвище великими літерами.

3.3. Рядкові перетворення.Напишіть програму, яка зчитує рядок і:
- виводить рядок у зворотному порядку (через зріз `[::-1]`);
- перевіряє, чи є рядок паліндромом (читається однаково з обох боків, без урахування регістру і пробілів);
- замінює всі пробіли на символ підкреслення `_` ;
- виводить кожен символ рядка на окремому рядку разом з його Unicode-кодом.

3.4. Аналіз тексту.Зчитайте довільне речення (кілька слів) і виконайте його аналіз:
- підрахуйте кількість слів у реченні;
- підрахуйте кількість голосних та приголосних літер (голосні: а е є и і ї о у ю я);
- знайдіть найдовше та найкоротше слово (якщо таких кілька — виведіть перше);
- виведіть слова у зворотному порядку;
- виведіть речення, де кожне слово починається з великої літери (`.title()`).

3.5. Валідація та форматування даних.Напишіть програму для перевірки та форматування введених даних:
- зчитайте номер телефону у довільному форматі (наприклад: `+38 (067) 123-45-67` або `0671234567`);
- видаліть усі нецифрові символи (`.replace()` або перебір), залишивши лише цифри;
- перевірте, чи отриманий рядок цифр має правильну довжину (12 цифр з кодом країни або 10 без);
- Виведіть номер у стандартному форматі: `+38 (XXX) XXX-XX-XX`.

3.6. Шифрування тексту. Реалізуйте простий шифр Цезаря:
- зчитайте рядок (лише латинські літери та пробіли) та ціле число — зсув (від 1 до 25);
- зашифруйте рядок: кожну літеру замініть на літеру, що стоїть через зсув позицій в алфавіті (a→d при зсуві 3). Пробіли не шифруються. Регістр зберігається.;
- виведіть зашифрований рядок;
- розшифруйте його (зсув у зворотній бік) і перевірте, що отримали оригінал.

Підказка: використайте `ord()`, `chr()` та операцію взяття залишку `%`.

3.7. Комплексний аналіз та форматування тексту. Зчитайте багаторядковий текст (кілька речень, введення завершується порожнім рядком) і виконайте повний аналіз:

- загальна статистика: кількість рядків, слів, символів (з пробілами і без);
- частотний аналіз: знайдіть 5 найуживаніших слів (без урахування регістру). Слова очищайте від розділових знаків (`.strip('.,!?;:')`);
- знайдіть усі унікальні слова та виведіть їх у алфавітному порядку (`sorted()`);
- перевірте кожне слово: скільки слів є паліндромами;
- сформуйте "резюме" тексту: перші 100 символів з трьома крапками в кінці, якщо текст довший;
- виведіть всю статистику у вигляді акуратно відформатованої таблиці (використайте `.center()`, `.ljust()`, `.rjust()`).

Контрольні запитання

1. Що означає «незмінність» рядків у Python? Що відбувається при виклику методу, наприклад `.upper()` — чи змінюється оригінальний рядок?

2. Поясніть різницю між від'ємними та додатними індексами рядка. Яке значення має індекс `-1`?

3. Що таке зріз (slice)? Запишіть вираз, який повертає останні 4 символи рядка s та вираз, що реверсує рядок.

4. Чим відрізняється метод `.find()` від `.index()`? Коли слід використовувати кожен з них?

5. Що повертає метод `.split()` без аргументів? Чим відрізняється `.split(' ')` від `.split()`?

6. Як об'єднати список рядків `['a','b','c']` в один рядок `'a-b-c'`? Запишіть відповідний вираз.

7. Поясніть різницю між `.strip()`, `.lstrip()` та `.rstrip()`. Наведіть приклад використання `.strip()` для очищення введених користувачем даних.

8. Для чого використовуються функції `ord()` та `chr()`? Наведіть приклад.

9. Що виведе вираз `'Python'[1:4:1]`? А `'Python'[::-1]`? Поясніть.

10. Як перевірити, що рядок складається виключно з цифр? Назвіть щонайменше два способи.

Лабораторна робота №4.

Списки

Мета роботи: Вивчити тип даних list у Python, способи створення, індексування, зрізи та базові операції зі списками; ознайомитися з вбудованими методами списків для додавання, видалення, пошуку та впорядкування елементів, а також зрозуміти різницю між методами, що змінюють список, і функціями, що повертають новий; опанувати техніку спискових включень (list comprehensions) як компактний і ефективний спосіб створення та перетворення списків, навчитися застосовувати їх у практичних задачах обробки даних.

Теоретичні відомості:

Список (list) — це впорядкована змінювана колекція елементів довільних типів. На відміну від рядків, списки є змінюваними: їхні елементи можна додавати, видаляти і змінювати після створення.

```
# Порожній список
empty = []
also_empty = list()

# Список цілих чисел
numbers = [3, 1, 4, 1, 5, 9, 2, 6]

# Список рядків
fruits = ['яблуко', 'груша', 'банан', 'вишня']

# Різнотипний список — дозволено!
mixed = [42, 3.14, 'текст', True, None]

# Вкладені списки (матриця 2×3)
matrix = [[1, 2, 3],
          [4, 5, 6]]

# Список із функції list()
chars = list('Python')   # ['P','y','t','h','o','n']
r     = list(range(5))   # [0, 1, 2, 3, 4]

# Основні властивості
print(len(numbers))      # 8  — кількість елементів
print(type(numbers))     # <class 'list'>
```

Списки підтримують ті самі принципи індексування та зрізів, що й рядки. Позитивні індекси рахуються з початку (від 0), від'ємні — з кінця (від -1).

```
lst = [10, 20, 30, 40, 50, 60]
#      0   1   2   3   4   5   — прямі індекси
#     -6  -5  -4  -3  -2  -1   — зворотні індекси

# Звернення до одного елемента
```

```
print(lst[0])       # 10
print(lst[3])       # 40
print(lst[-1])      # 60  — останній
print(lst[-2])      # 50  — передостанній

# Зрізи: список[start : stop : step]
print(lst[1:4])     # [20, 30, 40]
print(lst[:3])      # [10, 20, 30]  — від початку
print(lst[3:])      # [40, 50, 60]  — до кінця
print(lst[::2])     # [10, 30, 50]  — кожен другий
print(lst[::-1])    # [60, 50, 40, 30, 20, 10] — реверс

# Зміна елемента за індексом (списки змінювані!)
lst[2] = 99
print(lst)          # [10, 20, 99, 40, 50, 60]

# Зміна зрізу
lst[1:3] = [200, 300]
print(lst)          # [10, 200, 300, 40, 50, 60]

# Копія списку через зріз (не псевдонім!)
copy = lst[:]
```

Базові операції зі списками

```
a = [1, 2, 3]
b = [4, 5, 6]

# Конкатенація — оператор +
c = a + b
print(c)              # [1, 2, 3, 4, 5, 6]

# Повторення — оператор *
print([0] * 5)        # [0, 0, 0, 0, 0]
print([1, 2] * 3)     # [1, 2, 1, 2, 1, 2]

# Перевірка входження — in / not in
print(3 in a)         # True
print(7 not in a)     # True

# Вбудовані функції для числових списків
nums = [5, 2, 8, 1, 9, 3]
print(min(nums))      # 1
print(max(nums))      # 9
print(sum(nums))      # 28
print(len(nums))      # 6

# Перебір циклом for
for item in fruits:
    print(item, end=' ')

# Перебір з індексом
for i, item in enumerate(fruits):
```

```
    print(f'{i}: {item}')

# Перебір двох списків одночасно — zip()
names  = ['Іван', 'Марія', 'Олег']
scores = [88, 95, 72]
for name, score in zip(names, scores):
    print(f'{name}: {score}')
```

Методи додавання та видалення елементів

Метод / синтаксис	Опис та приклад
`lst.append(x)`	Додає `x` у кінець: `[1,2].append(3)` → `[1,2,3]`
`lst.insert(i, x)`	Вставляє `x` на позицію `i`: `[1,3].insert(1,2)` → `[1,2,3]`
`lst.extend(iter)`	Розширює список елементами iterable (як `+=`)
`lst.remove(x)`	Видаляє перше входження `x`; ValueError якщо немає
`lst.pop()`	Видаляє і повертає останній елемент
`lst.pop(i)`	Видаляє і повертає елемент за індексом `i`
`del lst[i]`	Видаляє елемент за індексом (оператор, не метод)
`del lst[i:j]`	Видаляє зріз зі списку
`lst.clear()`	Видаляє всі елементи: lst стає []

```
lst = [1, 2, 3]

# append — додати в кінець
lst.append(4)
print(lst)          # [1, 2, 3, 4]

# insert — вставити на позицію
lst.insert(0, 0)    # вставити 0 на початок
print(lst)          # [0, 1, 2, 3, 4]

# extend — додати кілька елементів
lst.extend([5, 6])
print(lst)          # [0, 1, 2, 3, 4, 5, 6]

# remove — видалити за значенням
lst.remove(0)
print(lst)          # [1, 2, 3, 4, 5, 6]

# pop() — взяти останній
last = lst.pop()
print(last, lst)    # 6  [1, 2, 3, 4, 5]

# pop(i) — взяти за індексом
second = lst.pop(1)
print(second, lst)  # 2  [1, 3, 4, 5]

# del — видалити зріз
del lst[1:3]
print(lst)          # [1, 5]
```

Методи пошуку та отримання інформації

Метод / синтаксис	Опис та приклад
`lst.index(x)`	Індекс першого входження `x`; ValueError якщо немає
`lst.index(x, s, e)`	Пошук x починаючи з позиції `s` до `e`
`lst.count(x)`	Кількість входжень x: `[1,2,1,3].count(1)` → `2`
`x in lst`	`True` якщо `x` є у списку (швидка перевірка)
`lst.copy()`	Поверхнева копія списку (аналог `lst[:]`)

```python
grades = [85, 92, 78, 92, 88, 92, 65]

# count — скільки разів зустрічається
print(grades.count(92))         # 3

# index — де знаходиться вперше
print(grades.index(92))         # 1

# index з діапазоном пошуку
print(grades.index(92, 2))      # 3  — пошук з позиції 2

# Безпечний пошук (уникаємо ValueError)
target = 100
if target in grades:
    print(grades.index(target))
else:
    print(f'{target} у списку відсутнє')

# Позиції ВСІХ входжень значення
positions = [i for i, g in enumerate(grades) if g == 92]
print(positions)                 # [1, 3, 5]
```

Python надає два інструменти сортування: метод `.sort()` та вбудовану функцію sorted(). Ключова різниця — у тому, чи змінюється оригінальний список.

Синтаксис	Повертає	Змінює список?	Коли використовувати
`list.sort()`	None	Так	Коли оригінальний порядок не потрібен
`sorted(list)`	Новий список	Ні	Коли потрібно зберегти оригінал
`list.sort(reverse=True)`	None	Так	Сортування у зворотньому порядку
`sorted(list, key=fn)`	Новий список	Ні	Сортування за довільним критерієм

```python
nums = [5, 2, 8, 1, 9, 3]

# sort() — сортує список НА МІСЦІ, повертає None
nums.sort()
print(nums)             # [1, 2, 3, 5, 8, 9]

nums.sort(reverse=True)
```

```
print(nums)             # [9, 8, 5, 3, 2, 1]

# sorted() — повертає НОВИЙ список, оригінал незмінний
original = [3, 1, 4, 1, 5]
new_list = sorted(original)
print(original)         # [3, 1, 4, 1, 5]  — незмінний
print(new_list)         # [1, 1, 3, 4, 5]

# Сортування рядків за довжиною — параметр key
words = ['банан', 'ківі', 'полуниця', 'яблуко']
words.sort(key=len)     # сортування за len() кожного слова
print(words)   # ['ківі', 'банан', 'яблуко', 'полуниця']

# Сортування без урахування регістру
cities = ['Харків', 'київ', 'Одеса', 'львів']
cities.sort(key=str.lower)
print(cities)  # ['Харків', 'київ', 'львів', 'Одеса']

# sorted() зі складним key: за другим елементом пари
students = [('Іван', 88), ('Марія', 95), ('Олег', 72)]
by_score = sorted(students, key=lambda s: s[1], reverse=True)
print(by_score)  # [('Марія', 95), ('Іван', 88), ('Олег', 72)]

# reverse() — реверс БЕЗ СОРТУВАННЯ
lst = [1, 2, 3, 4, 5]
lst.reverse()
print(lst)              # [5, 4, 3, 2, 1]
```

Списокове включення — це компактний синтаксис для створення нового списку на основі іншої послідовності. Воно замінює цикл `for + append()` і читається як «математичне» формулювання.

```
# ── Базова форма: без умови ──

# Звичайний цикл:
squares = []
for x in range(1, 6):
    squares.append(x ** 2)

# Еквівалентне списокове включення:
squares = [x ** 2 for x in range(1, 6)]
print(squares)          # [1, 4, 9, 16, 25]

# Перетворення типів
str_nums = ['1', '2', '3', '4', '5']
int_nums = [int(s) for s in str_nums]
print(int_nums)         # [1, 2, 3, 4, 5]

# Великі літери кожного слова
words = ['hello', 'world', 'python']
upper = [w.upper() for w in words]
print(upper)            # ['HELLO', 'WORLD', 'PYTHON']
```

```python
# ── З умовою (фільтрація) ──

nums = [1, -3, 4, -2, 7, -1, 5]

# Тільки додатні числа
positive = [x for x in nums if x > 0]
print(positive)          # [1, 4, 7, 5]

# Парні числа від 0 до 19
evens = [x for x in range(20) if x % 2 == 0]
print(evens)             # [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

# Слова довжиною більше 4 символів
words = ['ківі', 'полуниця', 'груша', 'ананас', 'іва']
long_words = [w for w in words if len(w) > 4]
print(long_words)        # ['полуниця', 'груша', 'ананас']

# ── З умовою у виразі (тернарний оператор) ──

# Замінити від'ємні числа нулем
data = [3, -1, 5, -7, 2]
non_neg = [x if x >= 0 else 0 for x in data]
print(non_neg)           # [3, 0, 5, 0, 2]

# ── Вкладені включення ──

# Таблиця множення у вигляді матриці
table = [[i * j for j in range(1, 5)] for i in range(1, 5)]
for row in table:
    print(row)
# [1, 2, 3, 4]
# [2, 4, 6, 8]
# ...

# Сплощення вкладеного списку
nested = [[1, 2], [3, 4], [5, 6]]
flat = [x for sublist in nested for x in sublist]
print(flat)              # [1, 2, 3, 4, 5, 6]
```

Важливо розуміти різницю між псевдонімом (alias) і копією. Присвоєння `b = a` не створює новий список — обидві змінні вказують на той самий об'єкт у пам'яті.

```python
# ── ПОМИЛКА: псевдонім ──
a = [1, 2, 3]
b = a                # b — це НЕ копія, а псевдонім!
b.append(4)
print(a)             # [1, 2, 3, 4]  — a теж змінився!

# ── Правильно: три способи створити копію ──

# 1. Зріз
c = a[:]
```

```python
# 2. Метод .copy()
d = a.copy()

# 3. Функція list()
e = list(a)

e.append(99)
print(a)          # [1, 2, 3, 4]  — оригінал незмінний
print(e)          # [1, 2, 3, 4, 99]

# УВАГА: .copy() — поверхнева копія!
# При вкладених списках вкладені об'єкти лишаються спільними.
# Для глибокого копіювання використайте copy.deepcopy()
import copy
matrix = [[1, 2], [3, 4]]
deep = copy.deepcopy(matrix)
deep[0][0] = 99
print(matrix)     # [[1, 2], [3, 4]] — незмінний
```

Корисні вбудовані функції для списків

Метод / синтаксис	Опис та приклад
`len(lst)`	Кількість елементів: `len([1,2,3]) → 3`
`sum(lst)`	Сума числових елементів: `sum([1,2,3]) → 6`
`min(lst)`	Мінімальний елемент: `min([3,1,2]) → 1`
`max(lst)`	Максимальний елемент: `max([3,1,2]) → 3`
`sorted(lst)`	Повертає новий відсортований список
`reversed(lst)`	Повертає ітератор у зворотному порядку
`enumerate(lst)`	Повертає пари (індекс, значення)
`zip(l1, l2)`	Об'єднує два списки в список пар
`list(range(n))`	Список з n елементів: `list(range(5)) → [0..4]`
`any(lst)`	`True` якщо хоча б один елемент True/не 0
`all(lst)`	`True` якщо ВСІ елементи True/не 0

```python
nums = [4, 7, 2, 9, 1, 6]

print(any(x > 8 for x in nums))   # True (9 > 8)
print(all(x > 0 for x in nums))   # True (всі > 0)
print(all(x > 5 for x in nums))   # False

# Індекс мінімального та максимального елемента
print(nums.index(min(nums)))   # 4  (значення 1)
print(nums.index(max(nums)))   # 3  (значення 9)

# reversed()
for x in reversed(nums):
    print(x, end=' ')          # 6 1 9 2 7 4
```

Завдання на лабораторну роботу

4.1. Операції зі списком. Створіть список із 7 довільних цілих чисел, введених користувачем (кожне число окремо через `input()`). Виведіть:

- Весь список та його довжину;
- Перший та останній елементи;
- Три перші та три останні елементи (через зрізи);
- Список у зворотньому порядку (через зріз [::-1]);
- Суму, мінімум та максимум елементів.

4.2. Методи додавання та видалення. Починаючи з порожнього списку, виконайте послідовно:

- додайте числа 10, 20, 30, 40, 50 по одному за допомогою `.append()`;
- вставте значення 15 між 10 і 20 за допомогою `.insert()`;
- розширте список значеннями [60, 70, 80] за допомогою `.extend()`;
- видаліть значення 30 методом `.remove()`;
- видаліть та виведіть останній елемент за допомогою `.pop()`;
- виведіть список після кожної операції.

4.3. Пошук та сортування. Задайте список оцінок студентів: [78, 92, 65, 88, 92, 71, 95, 88, 60, 92]. Виведіть:

- кількість оцінок 92 у списку (`.count()`);
- позицію першого входження 92 (`.index()`);
- список оцінок у зростаючому та спадаючому порядку (`sorted()`);
- оригінальний список (щоб показати, що `sorted()` не змінює його);
- відсортований список на місці через `.sort()`, виведіть результат.

4.4. Спискові включення. Використовуючи виключно спискові включення (без циклів for з append), створіть такі списки:

- квадрати чисел від 1 до 10;
- куби непарних чисел від 1 до 20;
- усі числа від 1 до 100, кратні одночасно 3 і 7;
- довжини слів із речення, введеного користувачем (`sentence.split()`);
- список пар (число, квадрат) для чисел від 1 до 8: [(1,1),(2,4),...,(8,64)];
- для кожного пункту виведіть результуючий список.

4.5. Статистика списку. Зчитайте від користувача N дійсних чисел (N вводиться першим). Обчисліть та виведіть:

- суму та середнє арифметичне;
- медіану (середній елемент відсортованого списку);
- кількість елементів, що перевищують середнє;
- список відсортований, з виділеними елементами вище середнього (додайте * перед числом);
- нормалізований список: кожен елемент ділиться на максимум (результат — числа від 0 до 1, округлені до 3 знаків після коми);
- використайте спискові включення там, де це доречно.

4.6. Обробка матриці. Створіть матрицю 4×4 (список списків) із довільними цілими числами

(введіть або задайте у коді). Виведіть:

- матрицю у вигляді таблиці (кожен рядок з новою стрічки, числа вирівняні);
- суму кожного рядка та суму кожного стовпця;
- головну діагональ матриці;
- транспоновану матрицю (рядки і стовпці міняються місцями) — через спискові включення;
- елементи матриці в один «плоский» список (сплощення через спискове включення).

4.7. Система управління оцінками. Розробіть програму для аналізу успішності групи студентів. Дані задаються у вигляді списку кортежів: `(прізвище, [оцінка1, оцінка2, ..., оцінка5])`. Визначте мінімум 5 студентів самостійно.
Програма повинна виконати:

- обчислити середній бал кожного студента (округлити до 2 знаків);
- відсортувати студентів за середнім балом (спадання) і вивести рейтинг;
- знайти «відмінників» (середній бал ≥ 90) та «відстаючих» (середній бал < 60) — через спискові включення;
- визначити найвищий і найнижчий одиночний бал серед усіх оцінок всіх студентів (сплощення через спискове включення);
- вивести «зведену відомість» — акуратну таблицю з прізвищем, оцінками та середнім балом (використайте f-рядки з вирівнюванням);
- обчислити загальний середній бал групи та відсоток студентів із балом вище за нього.

Контрольні запитання

1. Що таке список у Python? Чим він відрізняється від рядка з точки зору змінюваності?
2. Поясніть різницю між операціями `lst.append(x)` та `lst.extend([x, y])`. Наведіть приклад кожної.
3. Що повертає метод `lst.pop()`? Чим він відрізняється від `lst.remove(x)`?
4. Що таке псевдонім (alias) списку? Чому `b = a` не створює копію та як правильно скопіювати список?
5. Поясніть різницю між `lst.sort()` та `sorted(lst)`. Коли доцільно використовувати кожен варіант?
6. Що таке параметр `key` у методі `sort()`? Наведіть приклад сортування списку рядків за їхньою довжиною.
7. Запишіть спискове включення, яке повертає список квадратів непарних чисел від 1 до 20. Коли варто використовувати спискові включення замість звичайного циклу?
8. Що повернуть вирази `any([0, '', False])` та `all([1, 'a', True])`? Поясніть роботу цих функцій.
9. Як отримати індекс мінімального елемента списку nums? Запишіть відповідний вираз.
10. Що таке «сплощення» вкладеного списку? Запишіть спискове включення, яке перетворює [[1,2],[3,4],[5,6]] на [1,2,3,4,5,6].

Лабораторна робота №5.

Функції

Мета роботи: вивчити синтаксис визначення та виклику функцій у Python, види параметрів; засвоїти концепцію областей видимості змінних (LEGB-правило); ознайомитись з техніками функціонального програмування у Python, лямбда-функціями, замиканнями (closures) та декораторами.

Теоретичні відомості:

Функція — це іменований блок коду, який виконує певну задачу і може бути викликаний багаторазово. Функції дозволяють уникнути дублювання коду, розбити програму на логічні частини та полегшити тестування.

```
# Найпростіша функція без параметрів і без повернення
def greet():
    print('Привіт, світ!')

greet()             # Виклик функції
greet()             # Можна викликати скільки завгодно разів

# Функція з параметром
def greet_user(name):
    print(f'Привіт, {name}!')

greet_user('Олена')
greet_user('Іван')

# Функція з поверненням результату (return)
def square(x):
    return x ** 2

result = square(7)
print(result)       # 49
print(square(5))    # 25  — можна прямо у print()

# Функція є об'єктом — її можна присвоїти змінній
func = square
print(func(4))      # 16
```

Зверніть увагу!

Функція без явного `return` *повертає* `None`.

```
def f():
    print('ok')

result = f()
print(result)
```

```
ok
None
```

Тому завжди явно пишіть `return`, якщо функція має щось повертати.

Параметри — це імена у визначенні функції. Аргументи — це значення, що передаються при виклику. Python підтримує кілька видів параметрів.

Позиційні та іменовані аргументи

```
def describe_pet(name, animal, age):
    print(f'{name} — це {animal}, вік {age} років')
# Позиційні — порядок має значення
describe_pet('Барсик', 'кіт', 3)

# Іменовані (keyword) — порядок не важливий
describe_pet(age=5, name='Рекс', animal='пес')

# Можна змішувати: позиційні ЗАВЖДИ йдуть першими
describe_pet('Чоко', age=2, animal='папуга')
```

Значення за замовчуванням (default values)

```
# Параметр зі значенням за замовчуванням
def power(base, exp=2):           # exp=2 за замовчуванням
    return base ** exp

print(power(5))       # 25   (exp=2 за замовчуванням)
print(power(5, 3))    # 125  (exp=3 — перевизначено)
print(power(2, 10))   # 1024

# Реальний приклад: функція вітання з мовою за замовчуванням
def welcome(name, lang='uk'):
    messages = {
        'uk': f'Привіт, {name}!',
        'en': f'Hello, {name}!',
        'de': f'Hallo, {name}!',
    }
    return messages.get(lang, f'Hi, {name}!')

print(welcome('Олена'))           # Привіт, Олена!
print(welcome('Anna', 'en'))      # Hello, Anna!
```

Поширена помилка!

НІКОЛИ не використовуйте змінюваний об'єкт як значення за замовчуванням!

```
def bad(lst=[]):
     lst.append(1)
     return lst   # НЕБЕЗПЕЧНО!
bad()  # [1]       bad()  # [1, 1]  — об'єкт спільний для всіх викликів!
bad()
bad()
bad()
print(bad())
```

```
[1, 1, 1, 1, 1]
```

```python
# ПРАВИЛЬНО!
def good(lst=None):
     if lst is None: lst = []
```

*args — довільна кількість позиційних аргументів

```python
# *args збирає всі зайві позиційні аргументи у кортеж
def total(*args):
    print(f'Отримано аргументів: {len(args)}')
    print(f'Значення: {args}')
    return sum(args)

print(total(1, 2, 3))          # 6
print(total(10, 20, 30, 40))   # 100
print(total(5))                # 5

# Змішування: фіксований параметр + *args
def log(level, *messages):
    for msg in messages:
        print(f'[{level}] {msg}')

log('INFO', 'Програма запущена', 'Читання файлу')
log('ERROR', 'Щось пішло не так')
```

**kwargs — довільна кількість іменованих аргументів

```python
# **kwargs збирає іменовані аргументи у словник
def show_info(**kwargs):
    for key, value in kwargs.items():
        print(f'  {key}: {value}')

show_info(name='Іван', age=21, city='Київ')
# name: Іван
# age: 21
# city: Київ

# Практичний приклад: функція створення HTML-тегу
def make_tag(tag, content, **attrs):
    attr_str = ' '.join(f'{k}="{v}"' for k, v in attrs.items())
    if attr_str:
        return f'<{tag} {attr_str}>{content}</{tag}>'
    return f'<{tag}>{content}</{tag}>'

print(make_tag('p', 'Текст'))
# <p>Текст</p>
print(make_tag('a', 'Клік', href='https://example.com', class_='link'))
# <a href="https://example.com" class_="link">Клік</a>
```

Повний порядок параметрів

```python
# Правило: позиційні → *args → keyword-only → **kwargs
def full_example(pos1, pos2, *args, kw_only, **kwargs):
    print(f'Позиційні: {pos1}, {pos2}')
    print(f'*args: {args}')
    print(f'kw_only: {kw_only}')
    print(f'**kwargs: {kwargs}')

full_example(1, 2, 3, 4, 5, kw_only='обов\'язковий', extra=99)
# Позиційні: 1, 2
# *args: (3, 4, 5)
# kw_only: обов'язковий
# **kwargs: {'extra': 99}

# Розпакування при виклику: * і **
nums = [3, 7]
print(power(*nums))        # = power(3, 7)  → 2187

params = {'name': 'Марко', 'lang': 'en'}
print(welcome(**params))   # = welcome(name='Марко', lang='en')
```

Функція може повернути будь-який об'єкт Python. Якщо потрібно повернути кілька значень, Python автоматично упаковує їх у кортеж.

```python
# Повернення одного значення
def circle_area(r):
    import math
    return math.pi * r ** 2

# Повернення кількох значень (насправді — кортеж)
def circle_info(r):
    import math
    area      = math.pi * r ** 2
    perimeter = 2 * math.pi * r
    return area, perimeter          # повертає кортеж (area, perimeter)

a, p = circle_info(5)               # розпакування кортежу
print(f'Площа: {a:.2f}')
print(f'Периметр: {p:.2f}')

result = circle_info(5)             # або зберегти як кортеж
print(result)                       # (78.53..., 31.41...)

# Ранній вихід з функції через return
def safe_divide(a, b):
    if b == 0:
        return None                 # Ранній вихід
    return a / b

print(safe_divide(10, 2))  # 5.0
print(safe_divide(10, 0))  # None
```

Python визначає значення імені за правилом LEGB: Python шукає змінну послідовно у чотирьох «шарах» від внутрішнього до зовнішнього і зупиняється на першому знайденому.

Синтаксис	Опис / Приклад
`L — Local`	Локальний блок: змінні, визначені всередині поточної функції
`E — Enclosing`	Охоплююча функція: змінні зовнішньої функції (при вкладенні)
`G — Global`	Модульний рівень: змінні, визначені на рівні файлу/модуля
`B — Built-in`	Вбудований простір: `len`, `print`, `range`, `int` та інші

```
x = 'глобальна'          # G — глобальна змінна

def outer():
    x = 'охоплююча'   # E — у зовнішній функції

    def inner():
        x = 'локальна' # L — локальна змінна inner()
        print(x)       # 'локальна'  (знайдено у L)

    inner()
    print(x)           # 'об\'ємлююча'  (знайдено у E)

outer()
print(x)               # 'глобальна'  (знайдено у G)
```

Ключове слово global дозволяє зробити змінну глобальною

```
counter = 0            # глобальна змінна

def increment():
    global counter     # оголошуємо: працюємо з глобальною
    counter += 1

def get_counter():
    return counter     # читати глобальну можна без global
increment()
increment()
increment()
print(get_counter())   # 3

# Без global — помилка UnboundLocalError!
def bad_increment():
    # counter += 1   # ПОМИЛКА: Python бачить присвоєння
    pass             # і вважає counter локальною (ще не ініціалізованою)
```

Ключове слово `nonlocal`

```
# nonlocal — для змінної охоплюючої (не глобальної) функції
def make_counter():
    count = 0

    def increment():
        nonlocal count     # змінюємо змінну з outer-функції
```

```
        count += 1
        return count

    return increment

counter = make_counter()
print(counter())   # 1
print(counter())   # 2
print(counter())   # 3

# Два незалежних лічильники
c1 = make_counter()
c2 = make_counter()
c1(); c1()         # c1 → 2
c2()               # c2 → 1  (незалежний!)
```

Docstring — рядок документації одразу після `def`. Він пояснює, що робить функція, і доступний через `help()` та атрибут `__doc__`.

```
def calculate_bmi(weight: float, height: float) -> float:
    '''
    Обчислює індекс маси тіла (ІМТ).

    Параметри:
        weight (float): вага у кілограмах
        height (float): зріст у метрах

    Повертає:
        float: значення ІМТ, округлене до 2 знаків

    Приклад:
        >>> calculate_bmi(70, 1.75)
        22.86
    '''
    return round(weight / height ** 2, 2)

# Анотації типів — підказки (не примусово!)
def greet(name: str, times: int = 1) -> str:
    return (f'Привіт, {name}! ' * times).strip()

print(greet('Іван', 3))
# Привіт, Іван! Привіт, Іван! Привіт, Іван!

# Виведення документації
help(calculate_bmi)
```

Рекурсія — це техніка, коли функція викликає сама себе. Обов'язкові умови: базовий випадок (умова зупинки) і крок наближення до нього.

```python
# Класичний приклад: факторіал
def factorial(n):
    if n == 0 or n == 1:    # базовий випадок
        return 1
    return n * factorial(n - 1)   # рекурсивний виклик

print(factorial(5))   # 120
print(factorial(10))  # 3628800

# Числа Фібоначчі
def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)

print([fib(i) for i in range(10)])
# [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

# Сума цифр числа рекурсивно
def digit_sum(n):
    n = abs(n)
    if n < 10:
        return n
    return n % 10 + digit_sum(n // 10)

print(digit_sum(12345))   # 15
```

Обмеження рекурсії!

Python обмежує глибину рекурсії до ~1000 викликів (`sys.getrecursionlimit()`*).*
Для великих n рекурсія Фібоначчі дуже повільна — краще використовуйте цикл. При надмірній глибині буде кинуто виняток `RecursionError`.

Лямбда (lambda) — це коротка анонімна функція в одному рядку. Вона може мати будь-яку кількість параметрів, але лише один вираз, результат якого автоматично повертається.

```python
# Звичайна функція і її лямбда-еквівалент
def square(x):
    return x ** 2

square_l = lambda x: x ** 2

print(square(7))      # 49
print(square_l(7))    # 49  — однаково

# Лямбда з кількома параметрами
add  = lambda x, y: x + y
full = lambda first, last: f'{first} {last}'

print(add(3, 4))              # 7
print(full('Іван', 'Коваль')) # Іван Коваль
```

```python
# Лямбда з умовою (тернарний оператор)
classify = lambda n: 'парне' if n % 2 == 0 else 'непарне'
print(classify(6))    # парне
print(classify(7))    # непарне
```

Лямбда у функціях вищого порядку. Найбільша цінність лямбда — використання як аргумент у функціях, що приймають іншу функцію: `sorted()`, `map()`, `filter()`, `reduce()`.

```python
# sorted() з лямбда як key
students = [('Марія', 95), ('Іван', 78), ('Олег', 88), ('Ліна', 95)]

# Сортування за балом (за спаданням)
by_score = sorted(students, key=lambda s: s[1], reverse=True)
print(by_score)
# [('Марія', 95), ('Ліна', 95), ('Олег', 88), ('Іван', 78)]

# Сортування за двома критеріями: спочатку за балом (спад), потім за ім'ям (зріст)
by_two = sorted(students, key=lambda s: (-s[1], s[0]))
print(by_two)
# [('Ліна', 95), ('Марія', 95), ('Олег', 88), ('Іван', 78)]

# map() — застосувати функцію до кожного елементу
nums = [1, 2, 3, 4, 5]
squares = list(map(lambda x: x ** 2, nums))
print(squares)        # [1, 4, 9, 16, 25]

# filter() — відфільтрувати елементи
evens = list(filter(lambda x: x % 2 == 0, range(10)))
print(evens)          # [0, 2, 4, 6, 8]

# reduce() — згортка списку в одне значення
from functools import reduce
product = reduce(lambda acc, x: acc * x, [1, 2, 3, 4, 5])
print(product)        # 120  (1*2*3*4*5)
```

Коли лямбда, а коли def?
Лямбда підходить для простих одноразових функцій (особливо як аргумент).
def підходить для складної логіки, рекурсії, докстрінгів, повторного використання.
PEP8 радить НЕ присвоювати лямбду змінній: краще написати повноцінний def.

Замикання (Closures) виникає, коли вкладена функція «запам'ятовує» змінні зовнішньої функції навіть після того, як зовнішня функція завершила виконання. Вкладена функція несе з собою своє оточення.
Три умови замикання: 1) вкладена функція; 2) вона посилається на змінну зовнішньої функції; 3) зовнішня функція повертає вкладену

```python
# Базовий приклад замикання
def make_multiplier(factor):
    '''Повертає функцію, що множить на factor'''
    def multiplier(x):                # замикається на factor
        return x * factor
    return multiplier                 # повертаємо функцію (без дужок!)

double = make_multiplier(2)
triple = make_multiplier(3)

print(double(5))   # 10
print(triple(5))   # 15
print(double(8))   # 16

# Кожне замикання зберігає СВОЄ значення factor
print(double.__closure__[0].cell_contents)  # 2
print(triple.__closure__[0].cell_contents)  # 3
```

```python
# Замикання як генератор функцій — практичний приклад
def make_power(exp):
    return lambda x: x ** exp

square = make_power(2)
cube   = make_power(3)
quad   = make_power(4)

print([square(i) for i in range(1, 6)])  # [1, 4, 9, 16, 25]
print([cube(i)   for i in range(1, 6)])  # [1, 8, 27, 64, 125]

# Замикання зі збереженням стану (аналог об'єкта)
def make_bank_account(initial=0):
    balance = [initial]               # список, бо nonlocal не потрібен

    def deposit(amount):
        balance[0] += amount
        return balance[0]

    def withdraw(amount):
        if amount > balance[0]:
            return 'Недостатньо коштів'
        balance[0] -= amount
        return balance[0]

    def get_balance():
        return balance[0]

    return deposit, withdraw, get_balance

dep, wdr, bal = make_bank_account(1000)
print(dep(500))    # 1500
print(wdr(200))    # 1300
print(bal())       # 1300
```

Декоратор — це функція, яка приймає іншу функцію як аргумент, розширює її поведінку і повертає нову (модифіковану) функцію. Декоратори не змінюють оригінальну функцію, а «огортають» її.

Базовий декоратор

```python
# Крок 1: функція-обгортка (wrapper)
def my_decorator(func):
    def wrapper(*args, **kwargs):    # приймає будь-які аргументи
        print(f'--- Виклик {func.__name__} ---')
        result = func(*args, **kwargs)  # виклик оригінальної функції
        print(f'--- Завершено ---')
        return result
    return wrapper

# Крок 2: застосування через @
@my_decorator
def add(a, b):
    return a + b

# Це еквівалентно:  add = my_decorator(add)

print(add(3, 4))
# --- Виклик add ---
# --- Завершено ---
# 7
```

Декоратор із збереженням метаданих

```python
from functools import wraps

def log_calls(func):
    @wraps(func)                # зберігає __name__, __doc__
    def wrapper(*args, **kwargs):
        args_repr = ', '.join(repr(a) for a in args)
        print(f'Виклик: {func.__name__}({args_repr})')
        result = func(*args, **kwargs)
        print(f'Результат: {result!r}')
        return result
    return wrapper

@log_calls
def multiply(x, y):
    '''Множить два числа'''
    return x * y

multiply(4, 7)
# Виклик: multiply(4, 7)
# Результат: 28

print(multiply.__name__)  # multiply  (не wrapper!)
print(multiply.__doc__)   # Множить два числа
```

Практичні декоратори

```python
import time
from functools import wraps

# ── Декоратор вимірювання часу виконання ──
def timer(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.perf_counter()
        result = func(*args, **kwargs)
        elapsed = time.perf_counter() - start
        print(f'{func.__name__} виконано за {elapsed:.4f} с')
        return result
    return wrapper

@timer
def slow_sum(n):
    return sum(range(n))

slow_sum(10_000_000)  # slow_sum виконано за 0.3xx с

# ── Декоратор кешування результатів ──
def memoize(func):
    cache = {}
    @wraps(func)
    def wrapper(*args):
        if args not in cache:
            cache[args] = func(*args)
        return cache[args]
    return wrapper

@memoize
def fib(n):
    if n <= 1: return n
    return fib(n-1) + fib(n-2)

print(fib(40))   # Блискавично! (без кешу — хвилини)
```

Декоратори з параметрами

```python
# Декоратор з параметром — потрібен ще один рівень вкладення
def repeat(times):
    '''Повторює виклик функції times разів'''
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            result = None
            for _ in range(times):
                result = func(*args, **kwargs)
            return result
        return wrapper
    return decorator
```

```python
@repeat(3)
def say_hello(name):
    print(f'Привіт, {name}!')

say_hello('Олена')
# Привіт, Олена!
# Привіт, Олена!
# Привіт, Олена!

# Кілька декораторів застосовуються знизу вгору
@log_calls
@timer
def compute(n):
    return sum(i**2 for i in range(n))

compute(1000)
# Спочатку timer огортає compute,
# потім log_calls огортає timer(compute)
```

У Python функції є об'єктами першого класу: їх можна зберігати у змінних, передавати як аргументи, повертати з інших функцій і зберігати у структурах даних.

```python
# Зберігання функцій у словнику — «диспетчер»
def add(a, b):      return a + b
def sub(a, b):      return a - b
def mul(a, b):      return a * b
def div(a, b):      return a / b if b != 0 else 'Ділення на 0'

operations = {'+': add, '-': sub, '*': mul, '/': div}

op = input('Операція (+,-,*,/): ')
a  = float(input('a: '))
b  = float(input('b: '))

if op in operations:
    print(f'Результат: {operations[op](a, b)}')
else:
    print('Невідома операція')

# Функція вищого порядку — приймає і повертає функцію
def apply_twice(func, value):
    return func(func(value))

print(apply_twice(lambda x: x * 2, 3))   # 12  (3→6→12)
print(apply_twice(lambda x: x + 5, 10))  # 20  (10→15→20)

# Конвеєр (pipeline) функцій
def pipeline(*funcs):
    def process(value):
        for f in funcs:
            value = f(value)
        return value
```

```
    return process

clean = pipeline(
    str.strip,
    str.lower,
    lambda s: s.replace(' ', '_')
)

print(clean('  Hello World  '))  # hello_world
```

Завдання на лабораторну роботу

5.1. Базові функції. Напишіть три функції:
- `is_even(n)` — повертає `True`, якщо `n` парне;
- `max_of_three(a, b, c)` — повертає найбільше з трьох чисел;
- `power(base, exp=2)` — піднесення до степеня (з параметром за замовчуванням).

Для кожної функції напишіть мінімум 3 тестових виклики з виведенням результатів.

5.2. Функції з *args. Реалізуйте наступні функції:
- `sum_list(numbers)` — повертає суму елементів списку довільної довжини;
- `stats(*numbers)` — повертає словник `{'min': ..., 'max': ..., 'sum': ..., 'avg': ...}` з усіх числових аргументів, що були передані в функцію;

Продемонструйте виклики з різною кількістю аргументів.

5.3. Рекурсивні функції. Реалізуйте такі рекурсивні функції:
- `power(base, exp)` — піднесення до степеня без використання **;
- `factorial(number)` — визначення факторіала числа `number`.

Для кожної функції продемонструйте не менше 3 викликів.

5.4. Лямбда і функції вищого порядку. Виконайте всі завдання, використовуючи виключно `lambda`, `map()`, `filter()`, `sorted()`:
- зі списку чисел від -10 до 10 відфільтруйте тільки ті, що кратні 3;
- перетворіть список температур в градусах Цельсія [0, 20, 37, 100] у градуси Кельвіна;
- відсортуйте список рядків за останньою літерою слова;
- зі списку словників `[{'name': 'Іван', 'age': 22}, ...]` відфільтруйте тих, кому `>= 21`, відсортуйте за іменем.

5.6. Комплексна задача: функціональна обробка даних. Дано список студентів у вигляді словників з полями `name`, `grades` (список оцінок від 0 до 100), `year` (рік навчання). Задайте мінімум 8 студентів різних курсів.
Реалізуйте такі функції (без використання зовнішніх бібліотек):
- `average(grades)` — середня оцінка з точністю до 2 знаків (рекурсивно або через reduce);
- `classify(avg)` — повертає категорію: 'відмінник' (>=90), 'хорошист' (>=75), 'задовільно' (>=60), 'незараховано' (<60)
- `get_top(students, n=3)` — повертає n найкращих студентів (через `sorted` + `lambda`)

- `group_by_year(students)` — повертає словник `{рік: [список студентів]}`
- `report(students)` — декорована `@timer` функція, що виводить повну статистику: загальне зведення, топ-3, розбивка по роках, розподіл за категоріями

Вся логіка обробки має використовувати `map()`, `filter()`, `sorted()`, `lambda` і замикання.

Контрольні запитання

1. Що таке функція у Python? Чому використання функцій покращує якість коду?
2. Яка різниця між параметром і аргументом? Поясніть на прикладі.
3. Що таке `*args` і `**kwargs`? Який порядок параметрів є обов'язковим у визначенні функції?
4. Що таке LEGB-правило? Опишіть кожен рівень і наведіть приклад, де Python знаходить змінну на кожному рівні.
5. Для чого потрібні ключові слова `global` та `nonlocal`? Чим вони відрізняються?
6. Що таке лямбда-функція? Коли доцільно використовувати `lambda`, а коли `def`? Наведіть приклад використання `lambda` у `sorted()`.
7. Що таке замикання (closure)? Назвіть три умови його виникнення та поясніть практичне застосування.
8. Що таке декоратор? Поясніть, що означає запис `@my_decorator` над визначенням функції.
9. Для чого потрібен `@wraps(func)` всередині декоратора? Що відбудеться, якщо його не використати?
10. Що таке функція вищого порядку? Наведіть три приклади вбудованих функцій Python, що є функціями вищого порядку.

Лабораторна робота №6.

Робота з файлами

Мета роботи: Навчитися відкривати, читати та записувати текстові і бінарні файли у Python, зрозуміти різницю між режимами відкриття файлів та освоїти менеджер контексту with як основний безпечний спосіб роботи з файлами. Ознайомитися з модулями стандартної бібліотеки os, os.path, pathlib та shutil для безпечного створення, копіювання, переміщення, перейменування файлів та директорій, а також навчитися обходити файлову систему.Засвоїти роботу з різними кодуваннями тексту, навчитися виконувати пошук рядків у файлах та застосовувати регулярні вирази (модуль re) для розбору і фільтрації текстових даних у файлах.

Теоретичні відомості:

Вбудована функція `open()` відкриває файл і повертає файловий об'єкт. Режим відкриття визначає, що дозволено робити з файлом: читати, писати або дописувати.

Синтаксис / Режим	Опис / Дія
`'r'`	Читання (read). Файл повинен існувати. Режим за замовчуванням.
`'w'`	Запис (write). Якщо файл існує — очищає його. Якщо ні — створює.
`'a'`	Дописування (append). Дані додаються в кінець. Файл створюється якщо відсутній.
`'x'`	Ексклюзивне створення. Кидає FileExistsError якщо файл вже є.
`'r+'`	Читання і запис (файл повинен існувати).
`'w+'`	Читання і запис (файл очищається або створюється).
`'b'`	Бінарний режим (додається до інших): 'rb', 'wb', 'ab'.
`'t'`	Текстовий режим (за замовчуванням, можна не вказувати).

```
# Базовий синтаксис open()
# open(file, mode='r', encoding=None, errors='strict')
# Читання текстового файлу
f = open('data.txt', 'r', encoding='utf-8')
content = f.read()
f.close()              # ОБОВ'ЯЗКОВО закривати вручну
# Запис у файл
f = open('output.txt', 'w', encoding='utf-8')
f.write('Привіт, файл!\n')
f.close()

# Проблема: якщо між open() і close() виникне виняток —
# файл не закриється, ресурс «протече».
# Рішення: менеджер контексту with (розглянуто далі).
```

Менеджер контексту `with` гарантує, що файл буде закрито автоматично — навіть якщо в тілі блоку виникне виняток. Це найбезпечніший і рекомендований спосіб роботи з файлами в Python.

```
# with автоматично викликає f.close() при виході з блоку
with open('notes.txt', 'w', encoding='utf-8') as f:
    f.write('Перший рядок\n')
    f.write('Другий рядок\n')
# Тут файл вже закрито — навіть якщо виникне виняток

# Читання файлу
with open('notes.txt', 'r', encoding='utf-8') as f:
    content = f.read()
print(content)

# Два файли в одному with (Python 3.10+: можна у дужках)
with (open('source.txt', 'r', encoding='utf-8') as src,
      open('copy.txt',   'w', encoding='utf-8') as dst):
    dst.write(src.read())

# Перевірка: після with файл закрито
with open('notes.txt') as f:
    pass
print(f.closed)    # True
```

Правило!
Завжди використовуйте `with open(...) as f` замість `open() + f.close()`
Це захищає від витоку файлових дескрипторів та пошкодження даних.
Ніколи не забувайте `encoding='utf-8'` для текстових файлів з кирилицею.

Методи читання файлів

Синтаксис / Режим	Опис / Дія
`f.read()`	Читає весь файл як один рядок (str). *Обережно з великими файлами!*
`f.read(n)`	Читає n символів (або байт у бінарному режимі).
`f.readline()`	Читає один рядок (включно з `'\n'` у кінці).
`f.readlines()`	Повертає список усіх рядків файлу.
`for ln in f:`	Ітерація по рядках — найефективніший спосіб для великих файлів.
`f.tell()`	Повертає поточну позицію (в байтах) у файлі.
`f.seek(n)`	Переміщує курсор на позицію n байт від початку.

```
# 1. read() — весь файл як рядок
with open('poem.txt', 'r', encoding='utf-8') as f:
    text = f.read()
print(f'Символів: {len(text)}')

# 2. readline() — рядок за рядком вручну
with open('poem.txt', 'r', encoding='utf-8') as f:
    first  = f.readline()   # перший рядок
```

```
    second = f.readline()   # другий рядок
    print(first.strip(), second.strip())

# 3. readlines() — список рядків
with open('poem.txt', 'r', encoding='utf-8') as f:
    lines = f.readlines()   # ['рядок1\n', 'рядок2\n', ...]
print(f'Рядків: {len(lines)}')
print(lines[0].strip())    # перший рядок без \n
# 4. Ітерація — НАЙКРАЩИЙ спосіб для великих файлів
# Зчитує по одному рядку, не завантажуючи весь файл у пам'ять
with open('poem.txt', 'r', encoding='utf-8') as f:
    for i, line in enumerate(f, start=1):
        print(f'{i:3}: {line}', end='')

# 5. seek() і tell()
with open('poem.txt', 'r', encoding='utf-8') as f:
    print(f.read(10))    # читаємо 10 символів
    print(f.tell())      # поточна позиція
    f.seek(0)            # повертаємось на початок
    print(f.read(10))    # читаємо ті ж 10 символів знову
```

Методи запису. Дописування у файл

```
# write() — записує рядок (без автоматичного \n!)
with open('log.txt', 'w', encoding='utf-8') as f:
    f.write('Рядок перший\n')
    f.write('Рядок другий\n')
    chars = f.write('Рядок третій\n')
    print(f'Записано символів: {chars}')  # write() повертає кількість

# writelines() — записує список рядків
lines = ['Яблуко\n', 'Груша\n', 'Банан\n']
with open('fruits.txt', 'w', encoding='utf-8') as f:
    f.writelines(lines)    # \n НЕ додається автоматично!

# Режим 'a' — ДОПИСУВАННЯ (append), не затирає файл
import datetime
now = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

with open('events.log', 'a', encoding='utf-8') as f:
    f.write(f'[{now}] Програма запущена\n')

# print() теж вміє писати у файл через параметр file=
with open('report.txt', 'w', encoding='utf-8') as f:
    print('Звіт', file=f)
    print('=' * 30, file=f)
    for i in range(1, 4):
        print(f'Запис {i}: {i**2}', file=f)
```

Кодування визначає, як символи Unicode перетворюються на байти при збереженні. Без явного зазначення encoding Python використовує системне кодування (у Windows це часто `cp1251`, у Linux/Mac — `utf-8`).

Синтаксис / Режим	Опис / Дія
`'utf-8'`	Універсальне. Підтримує всі мови світу. Рекомендовано завжди.
`'utf-8-sig'`	UTF-8 з BOM-маркером. Потрібен для Excel та деяких Windows-програм.
`'cp1251'`	Кирилиця Windows (Windows-1251). Часто у старих файлах.
`'latin-1'`	ISO 8859-1. Зустрічається у файлах із Заходу.
`errors='strict'`	За замовчуванням: кидає UnicodeDecodeError при помилці.
`errors='ignore'`	Ігнорує символи, які не вдається декодувати.
`errors='replace'`	Замінює проблемні символи на '?' або '�'.

```python
# Запис у різних кодуваннях
text = 'Привіт, це текст українською!'

# UTF-8 — рекомендовано
with open('utf8.txt', 'w', encoding='utf-8') as f:
    f.write(text)

# UTF-8 з BOM (для сумісності з Excel)
with open('utf8_bom.txt', 'w', encoding='utf-8-sig') as f:
    f.write(text)

# CP1251 — Windows-кирилиця
with open('cp1251.txt', 'w', encoding='cp1251') as f:
    f.write(text)

# Визначення кодування невідомого файлу (бібліотека chardet)
# pip install chardet
# import chardet
# with open('unknown.txt', 'rb') as f:
#     raw = f.read()
# detected = chardet.detect(raw)
# print(detected)   # {'encoding': 'UTF-8', 'confidence': 0.99}

# Безпечне читання з errors='replace'
with open('cp1251.txt', 'r', encoding='utf-8', errors='replace') as f:
    content = f.read()    # неправильні байти → '\ufffd'
print(content)

# Правильне читання cp1251-файлу
with open('cp1251.txt', 'r', encoding='cp1251') as f:
    content = f.read()    # тепер текст коректний
print(content)
```

Модуль `os` надає функції для взаємодії з операційною системою: керування файлами, директоріями та середовищем виконання. `os.path` — підмодуль для роботи зі шляхами файлів.

```python
import os

# ── Інформація про поточне оточення ──
print(os.getcwd())              # поточна директорія
print(os.listdir('.'))          # список файлів у поточній директорії
print(os.listdir('C:/Users'))   # список файлів у вказаній папці

# ── Робота з директоріями ──
os.mkdir('new_folder')          # створити папку
os.makedirs('a/b/c', exist_ok=True)  # створити вкладені папки
os.chdir('new_folder')          # перейти у папку
os.chdir('..')                  # повернутися на рівень вгору
os.rmdir('new_folder')          # видалити ПОРОЖНЮ папку

# ── Операції з файлами ──
os.rename('old.txt', 'new.txt') # перейменувати файл або папку
os.remove('temp.txt')           # видалити файл

# ── os.path: робота зі шляхами ──
path = '/home/user/projects/lab6/data.txt'

print(os.path.basename(path))   # data.txt
print(os.path.dirname(path))    # /home/user/projects/lab6
print(os.path.split(path))
# ('/home/user/projects/lab6', 'data.txt')

print(os.path.splitext('data.txt'))  # ('data', '.txt')
print(os.path.join('lab6', 'data', 'file.txt'))
# lab6/data/file.txt  (або lab6\data\file.txt на Windows)

print(os.path.exists('data.txt'))    # True/False
print(os.path.isfile('data.txt'))    # чи є файлом
print(os.path.isdir('lab6'))         # чи є директорією
print(os.path.getsize('data.txt'))   # розмір у байтах
print(os.path.abspath('data.txt'))   # абсолютний шлях
```

Модуль `pathlib` пропонує об'єктно-орієнтований інтерфейс до файлової системи. Path-об'єкти — більш зручна та читабельна альтернатива `os.path`.

```python
from pathlib import Path

# Створення Path-об'єкта
p = Path('.')                    # поточна директорія
p = Path('/home/user/projects')  # абсолютний шлях
p = Path.home()                  # домашня директорія користувача
p = Path.cwd()                   # поточна робоча директорія

# Оператор / для з'єднання частин шляху
project = Path('lab6')
data_file = project / 'data' / 'students.txt'
print(data_file)    # lab6/data/students.txt

# Властивості Path-об'єкта
p = Path('lab6/data/report.csv')
print(p.name)        # report.csv
print(p.stem)        # report
print(p.suffix)      # .csv
print(p.parent)      # lab6/data
print(p.parents[0])  # lab6/data
print(p.parents[1])  # lab6
print(p.is_absolute()) # False

# Перевірки та інформація
p = Path('data.txt')
print(p.exists())    # True / False
print(p.is_file())   # чи є файлом
print(p.is_dir())    # чи є директорією
print(p.stat().st_size)  # розмір у байтах

# Читання та запис через pathlib
p = Path('hello.txt')
p.write_text('Привіт, pathlib!', encoding='utf-8')
print(p.read_text(encoding='utf-8'))  # Привіт, pathlib!

# Перебір файлів у директорії
for f in Path('.').iterdir():
    if f.is_file():
        print(f.name, f.stat().st_size, 'bytes')

# glob() — пошук файлів за шаблоном
for txt in Path('.').glob('*.txt'):
    print(txt)

# rglob() — рекурсивний пошук у підпапках
for py in Path('.').rglob('*.py'):
    print(py)

# Створення директорій
Path('output/logs').mkdir(parents=True, exist_ok=True)
```

Модуль `shutil` (shell utilities) надає функції вищого рівня для операцій над файлами та директоріями. На відміну від `os`, він вміє рекурсивно копіювати та видаляти директорії.

Синтаксис / Режим	Опис / Дія
`shutil.copy(src, dst)`	Копіює файл src у dst (без метаданих).
`shutil.copy2(src, dst)`	Копіює файл зі збереженням метаданих (дата, атрибути).
`shutil.copytree(src, dst)`	Рекурсивно копіює директорію з усім вмістом.
`shutil.move(src, dst)`	Переміщує або перейменовує файл/директорію.
`shutil.rmtree(path)`	Рекурсивно видаляє директорію з усім вмістом.
`shutil.make_archive(...)`	Створює архів (zip, tar, gztar).
`shutil.unpack_archive(...)`	Розпаковує архів у вказану папку.
`shutil.disk_usage(path)`	Повертає загальний, використаний та вільний простір.

```python
import shutil
from pathlib import Path

# ── Копіювання файлів ──
shutil.copy('report.txt', 'backup/report.txt')   # копія
shutil.copy2('data.csv', 'archive/')             # зберігає метадані

# ── Безпечне копіювання з перевіркою ──
src = Path('data.txt')
dst = Path('backup/data.txt')

dst.parent.mkdir(parents=True, exist_ok=True)   # створимо папку якщо немає
if src.exists():
    shutil.copy2(src, dst)
    print(f'Скопійовано: {src} → {dst}')
else:
    print(f'Файл не знайдено: {src}')

# ── Копіювання директорії ──
shutil.copytree('project/', 'project_backup/',
                ignore=shutil.ignore_patterns('*.pyc', '__pycache__'))

# ── Переміщення і перейменування ──
shutil.move('old_name.txt', 'new_name.txt')     # перейменування
shutil.move('file.txt', 'archive/')             # переміщення у папку

# ── Видалення директорії рекурсивно ──
# УВАГА: операція незворотна!
tmp = Path('temp_folder')
if tmp.exists():
    shutil.rmtree(tmp)
    print('Тимчасову папку видалено')

# ── Архівування ──
shutil.make_archive('project_v1', 'zip', 'project/')   # → project_v1.zip
shutil.make_archive('backup', 'gztar', 'data/')        # → backup.tar.gz
```

```
# ── Розпакування ──
shutil.unpack_archive('project_v1.zip', 'restored/')

# ── Інформація про диск ──
usage = shutil.disk_usage('/')
print(f'Всього:    {usage.total / 1024**3:.1f} ГБ')
print(f'Вільно:    {usage.free  / 1024**3:.1f} ГБ')
print(f'Зайнято:   {usage.used  / 1024**3:.1f} ГБ')
```

Базовий пошук рядка у файлі можна виконати простою перевіркою `in`, але для складніших шаблонів використовують регулярні вирази (модуль re).

Простий пошук рядків

```
# ── Пошук рядків, що містять ключове слово ──
def find_lines(filename, keyword, encoding='utf-8'):
    '''Повертає список рядків, що містять keyword'''
    results = []
    with open(filename, 'r', encoding=encoding) as f:
        for lineno, line in enumerate(f, start=1):
            if keyword.lower() in line.lower():  # без урахування регістру
                results.append((lineno, line.rstrip()))
    return results

hits = find_lines('notes.txt', 'python')
for lineno, line in hits:
    print(f'Рядок {lineno:3}: {line}')

# ── Підрахунок слів у файлі ──
def word_count(filename, encoding='utf-8'):
    with open(filename, 'r', encoding=encoding) as f:
        text = f.read()
    words = text.split()
    unique = set(w.lower().strip('.,!?;:') for w in words)
    return {'total': len(words), 'unique': len(unique),
            'lines': text.count('\n') + 1,
            'chars': len(text)}

stats = word_count('essay.txt')
print(f'Слів всього:  {stats["total"]}')
print(f'Унікальних:   {stats["unique"]}')
print(f'Рядків:       {stats["lines"]}')
```

Регулярний вираз (`regex`) — це шаблон для пошуку та заміни тексту. Модуль re забезпечує потужний інструментарій для роботи з такими шаблонами.

Синтаксис / Режим	Опис / Дія
`re.search(pat, s)`	Перше входження pat у рядку s. Повертає Match або None.
`re.match(pat, s)`	Перевіряє pat лише на ПОЧАТКУ рядка.
`re.findall(pat, s)`	Список УСІХ входжень (рядки).
`re.finditer(pat, s)`	Ітератор Match-об'єктів усіх входжень.
`re.sub(pat, repl, s)`	Замінює всі входження pat на repl у рядку s.

`re.compile(pat)`	Компілює шаблон для багаторазового використання.
`re.ignorecase`	Прапор: пошук без урахування регістру (re.I).
`re.multiline`	Прапор: ^ та $ відповідають початку/кінцю кожного рядка.

Основні елементи шаблонів

```
#  .       – будь-який символ (крім \n)
#  \d      – цифра [0-9]
#  \D      – не цифра
#  \w      – літера, цифра або _  [a-zA-Z0-9_]
#  \s      – пробільний символ (пробіл, \t, \n)
#  ^       – початок рядка
#  $       – кінець рядка
#  *       – 0 або більше повторень
#  +       – 1 або більше повторень
#  ?       – 0 або 1 повторення
#  {n,m}   – від n до m повторень
#  [abc]   – один із символів a, b, c
#  [^abc]  – будь-який символ, крім a, b, c
#  (...)   – група (можна витягнути через .group())
#  |       – або (a|b – a або b)
```

Практичні приклади пошуку у файлах

```
import re
# ── Витягти всі email-адреси з файлу ──
email_pattern = re.compile(
    r'[a-zA-Z0-9._%+\-]+@[a-zA-Z0-9.\-]+\.[a-zA-Z]{2,}'
)

def extract_emails(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        text = f.read()
    return email_pattern.findall(text)

emails = extract_emails('contacts.txt')
print(f'Знайдено email: {len(emails)}')
for e in emails:
    print(f'  {e}')
# ── Витягти всі номери телефонів ──
phone_pattern = re.compile(
    r'(?:\+38)?[\s\-]?\(?0\d{2}\)?[\s\-]?\d{3}[\s\-]?\d{2}[\s\-]?\d{2}'
)

def extract_phones(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        text = f.read()
    return phone_pattern.findall(text)

# ── Знайти рядки, що відповідають шаблону ──
def grep(filename, pattern, flags=re.IGNORECASE):
    '''Аналог unix grep – знаходить рядки за regex-шаблоном'''
    pat = re.compile(pattern, flags)
```

```
    results = []
    with open(filename, 'r', encoding='utf-8') as f:
        for no, line in enumerate(f, 1):
            if pat.search(line):
                results.append((no, line.rstrip()))
    return results

# Знайти рядки, що містять дати формату DD.MM.YYYY
date_hits = grep('report.txt', r'\d{2}\.\d{2}\.\d{4}')
for no, line in date_hits:
    print(f'{no}: {line}')
# ── Заміна шаблонів у файлі ──
def replace_in_file(filename, pattern, replacement, out_file=None):
    '''Замінює всі входження pattern у файлі'''
    with open(filename, 'r', encoding='utf-8') as f:
        text = f.read()

    new_text, count = re.subn(pattern, replacement, text)
    print(f'Замінено входжень: {count}')
    target = out_file or filename
    with open(target, 'w', encoding='utf-8') as f:
        f.write(new_text)

# Замінити всі дати з DD.MM.YYYY на YYYY-MM-DD
replace_in_file(
    'report.txt',
    r'(\d{2})\.(\d{2})\.(\d{4})',   # групи: день, місяць, рік
    r'\3-\2-\1',                    # переставляємо групи
    out_file='report_fixed.txt'
)
# ── Парсинг CSV вручну через re ──
log_pattern = re.compile(
    r'^(\d{4}-\d{2}-\d{2})\s+(\w+)\s+(.+)$'
)

with open('app.log', 'r', encoding='utf-8') as f:
    for line in f:
        m = log_pattern.match(line)
        if m:
            date, level, msg = m.groups()
            if level == 'ERROR':
                print(f'[{date}] ПОМИЛКА: {msg}')
```

При роботі з файлами можуть виникати різні помилки. Їх необхідно обробляти, щоб програма не падала з некоректним повідомленням.

Синтаксис / Режим	Опис / Дія
`FileNotFoundError`	Файл або директорія не існує.
`PermissionError`	Немає прав доступу на читання або запис.
`IsADirectoryError`	Спроба відкрити директорію як файл.
`FileExistsError`	Файл вже існує (режим 'x').

`UnicodeDecodeError`	Файл не вдається декодувати в заданому кодуванні.
`OSError`	Базовий клас для всіх файлових помилок (ловить усі вище).

```python
from pathlib import Path

def safe_read(filepath, encoding='utf-8'):
    '''Безпечне читання файлу з обробкою помилок'''
    path = Path(filepath)
    try:
        with open(path, 'r', encoding=encoding) as f:
            return f.read()
    except FileNotFoundError:
        print(f'Файл не знайдено: {path}')
    except PermissionError:
        print(f'Немає прав доступу: {path}')
    except UnicodeDecodeError:
        print(f'Помилка кодування. Спробуйте encoding="cp1251"')
    except OSError as e:
        print(f'Помилка файлової системи: {e}')
    return None

def safe_write(filepath, content, encoding='utf-8'):
    '''Безпечний запис з резервною копією'''
    path = Path(filepath)
    backup = path.with_suffix('.bak')
    try:
        # Зробити резервну копію якщо файл існує
        if path.exists():
            import shutil
            shutil.copy2(path, backup)
        with open(path, 'w', encoding=encoding) as f:
            f.write(content)
        print(f'Записано успішно: {path}')
        return True
    except PermissionError:
        print(f'Немає прав запису: {path}')
    except OSError as e:
        print(f'Помилка запису: {e}')
        # Відновити з резервної копії
        if backup.exists():
            shutil.copy2(backup, path)
    return False

# Використання
content = safe_read('data.txt')
if content is not None:
    processed = content.upper()
    safe_write('data_upper.txt', processed)
```

Завдання на лабораторну роботу

Перед виконанням завдань створіть у своєму проекті папку lab6_data/ і підготуйте в ній текстові файли з тестовим вмістом (студенти, оцінки, довільний текст). Усі завдання виконуйте з цими файлами.

6.1. Читання і запис текстового файлу. Виконайте такі дії:
- вручну (не кодом!) створіть файл `students.txt` із 8–10 рядків у форматі: `"Прізвище Ім'я, оцінка"` (оцінка від 50 до 100);
- напишіть програму, яка читає файл рядок за рядком і виводить кожен рядок з номером;
- підрахуйте і виведіть кількість рядків, кількість слів (загалом у файлі), розмір файлу в байтах (через `os.path.getsize()`);
- запишіть у новий файл `students_upper.txt` вміст `students.txt` у верхньому регістрі — ОБОВ'ЯЗКОВО через `with` і обробку `UnicodeDecodeError`.

6.2. Ведення журналу подій. Реалізуйте систему логування:
- Напишіть функцію `log_event(filename, level, message)`, яка дописує у файл рядок формату: `[YYYY-MM-DD HH:MM:SS] [LEVEL] message`;
- використовуйте режим `'a'` і модуль `datetime` для отримання поточного часу;
- викличте функцію мінімум 10 разів з різними рівнями: `INFO`, `WARNING`, `ERROR`;
- напишіть функцію `read_log(filename, level=None)`, яка читає лог і повертає список записів — усіх (якщо `level=None`) або тільки заданого рівня;
- виведіть кількість записів кожного рівня.

6.3. Операції з файловою системою (`os` та `pathlib`). Напишіть програму, яка:
- створює структуру директорій: `lab6_project/src/`, `lab6_project/data/`, `lab6_project/output/` (через `pathlib.mkdir` з `exist_ok=True`);
- виводить дерево цієї структури (назву кожного елемента з відступом залежно від глибини);
- для кожного файлу у `lab6_data/` виводить: ім'я файлу, розширення, розмір у байтах і дату останньої зміни (через `Path.stat().st_mtime` та `datetime.fromtimestamp()`);
- використайте `glob('*.txt')` для пошуку тільки текстових файлів.

6.4. Безпечне копіювання та резервне збереження (`shutil`). Реалізуйте утиліту резервного копіювання:
- функція `backup_file(src_path)` — копіює файл у папку backup/ з додаванням часової мітки до імені: `file.txt → backup/file_2026-11-15_14-30.txt`;
- функція `backup_folder(src_dir, dst_dir)` — копіює директорію через `shutil.copytree`, ігноруючи `*.tmp` і `*.log` файли;
- функція `restore_latest(filename)` — знаходить найсвіжішу резервну копію (через `glob` і `sorted`) і відновлює її;
- Продемонструйте роботу всіх трьох функцій і обробіть можливі виняткові ситуації (FileNotFoundError, PermissionError)

6.5. Пошук і аналіз тексту за допомогою регулярних виразів. Створіть файл `contacts.txt` із 10–15 рядками у довільному форматі, де зустрічаються: email-адреси, номери телефонів (у

різних форматах), дати (DD.MM.YYYY).
Напишіть програму, яка:

- витягує всі email-адреси з файлу (шаблон: `user@domain.tld`);
- витягує всі номери телефонів (шаблон враховує різні формати: +38(067)..., 0671234567, 067-123-45-67);
- знаходить всі дати у форматі DD.MM.YYYY і конвертує їх у ISO-формат YYYY-MM-DD через `re.sub()` із групами;
- Записує структурований результат у `results.txt`: три секції зі знайденими email, телефонами і датами.

Контрольні запитання

1. Що таке менеджер контексту `with`? Чому він є кращою альтернативою до явного виклику `f.close()`?

2. Назвіть і поясніть 5 режимів відкриття файлів. Чим відрізняється режим `'w'` від `'a'`? Що станеться з існуючим файлом при відкритті в режимі `'w'`?

3. Яка різниця між `f.read()`, `f.readline()` і `f.readlines()`? Коли доцільно використовувати ітерацію по файловому об'єкту `for line in f`?

4. Що таке кодування тексту? Чому важливо явно вказувати `encoding='utf-8'` при відкритті файлів? Що станеться якщо цього не зробити на Windows?

5. Чим `pathlib.Path` відрізняється від `os.path`? Яке переваги дає оператор `/` для Path-об'єктів?

6. Назвіть три функції модуля `shutil` для роботи з файлами. Чим `shutil.copy()` відрізняється від `shutil.copy2()`?

7. Яка різниця між `os.remove()` та `shutil.rmtree()`? В яких ситуаціях потрібна кожна з них?

8. Що таке регулярний вираз? Поясніть значення символів \d, \w, \s, +, * та ? у шаблоні.

9. Яка різниця між `re.search()` та `re.match()`? Коли використовувати `re.findall()`, а коли `re.finditer()`?

10. Назвіть три винятки, які можуть виникнути при роботі з файлами. Як правильно їх обробляти за допомогою `try-except`?

Лабораторна робота №7.

Бібліотеки NumPy та Matplotlib

Мета роботи: Ознайомитися з бібліотекою NumPy. Навчитися створювати масиви, виконувати векторизовані математичні операції, розв'язувати задачі лінійної алгебри та статистики, які стануть у нагоді при вивченні фізики, математики та інженерних дисциплін. Навчитися будувати та форматувати графіки за допомогою бібліотеки Matplotlib. Сформувати практичні навички збереження графіків у файли різних форматів, а також застосовувати набуті знання для вирішення прикладних задач із фізики, математики та аналізу даних.

Теоретичні відомості:

Частина І. Бібліотека NumPy.

`NumPy` (Numerical Python) — фундаментальна бібліотека наукових обчислень у Python. Її основа — об'єкт `ndarray` (N-dimensional array): багатовимірний масив однорідних (одного типу) елементів.

Чому NumPy швидший за звичайні списки Python?
Елементи масиву зберігаються у неперервній ділянці пам'яті (на відміну від списків). Операції виконуються у скомпільованому C/Fortran-коді без інтерпретатора Python.

Векторизація: одна операція застосовується одночасно до всіх елементів

```python
import numpy as np

# ── Створення масивів ──

# Зі списку Python
a = np.array([1, 2, 3, 4, 5])
print(a)          # [1 2 3 4 5]
print(type(a))    # <class 'numpy.ndarray'>

# Двовимірний масив (матриця) зі вкладеного списку
m = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

# Ключові атрибути масиву
print(a.shape)     # (5,)     — форма (кортеж розмірностей)
print(m.shape)     # (3, 3)   — 3 рядки, 3 стовпці
print(m.ndim)      # 2        — кількість вимірів
print(m.size)      # 9        — загальна кількість елементів
print(m.dtype)     # int64    — тип даних
print(m.itemsize)  # 8        — байт на елемент
```

Функції створення масивів

Синтаксис	Опис / Результат
`np.zeros((m, n))`	Масив нулів форми m×n: `np.zeros((2,3))`
`np.ones((m, n))`	Масив одиниць: `np.ones((3,3))`
`np.full((m,n), val)`	Масив, заповнений `val`: `np.full((2,4), 7)`
`np.eye(n)`	Одинична матриця `n×n` (діагональ = 1)
`np.arange(s,e,step)`	Масив чисел у діапазоні з кроком (як `range()`)
`np.linspace(s,e,n)`	`n` рівномірно розміщених точок від `s` до `e` (включно)
`np.logspace(s,e,n)`	`n` точок у логарифмічній шкалі від `10^s` до `10^e`
`np.random.rand(m,n)`	Випадкові числа `[0,1)` рівномірного розподілу
`np.random.randn(m,n)`	Випадкові числа нормального розподілу `N(0,1)`
`np.random.randint(l,h,n)`	`n` випадкових цілих чисел від `l` до `h-1`

```
# arange — аналог range(), але повертає масив
x = np.arange(0, 10, 2)       # [0 2 4 6 8]
x = np.arange(0.0, 1.1, 0.25) # [0.   0.25 0.5  0.75 1.  ]

# linspace — рівномірна сітка точок (ідеальна для графіків!)
x = np.linspace(0, 2*np.pi, 100)  # 100 точок від 0 до 2π

# ones, zeros, eye
print(np.zeros((2, 4)))
# [[0. 0. 0. 0.]
#  [0. 0. 0. 0.]]

print(np.eye(3))
# [[1. 0. 0.]
#  [0. 1. 0.]
#  [0. 0. 1.]]

# Вказання типу даних
a = np.array([1, 2, 3], dtype=np.float64)
b = np.zeros(5, dtype=np.int32)
print(a.dtype)   # float64
```

Індексування, зрізи та форма масивів

```
import numpy as np

# ── Одновимірний масив ──
a = np.array([10, 20, 30, 40, 50])
print(a[0])      # 10  — перший елемент
print(a[-1])     # 50  — останній
print(a[1:4])    # [20 30 40]  — зріз
print(a[::2])    # [10 30 50]  — кожен другий

# ── Двовимірний масив (матриця) ──
m = np.array([[1, 2, 3],
              [4, 5, 6],
```

```
              [7, 8, 9]])

print(m[0, 0])   # 1  — рядок 0, стовпець 0
print(m[1, 2])   # 6  — рядок 1, стовпець 2
print(m[0])      # [1 2 3]  — весь перший рядок
print(m[:, 1])   # [2 5 8]  — весь другий стовпець
print(m[0:2, 1:3])
# [[2 3]
#  [5 6]]  — підматриця 2×2

# ── Булеве індексування ──
a = np.array([3, -1, 7, -4, 5, -2])
mask = a > 0
print(mask)      # [ True False  True False  True False]
print(a[mask])   # [3 7 5]  — тільки додатні елементи

# Скорочений запис
print(a[a > 0])  # [3 7 5]
print(a[a % 2 == 0])  # [-4 -2]  — тільки парні

# ── Зміна форми масиву ──
a = np.arange(12)        # [ 0  1  2 ... 11]
m = a.reshape(3, 4)      # матриця 3×4
v = m.flatten()          # назад у 1D
t = m.reshape(-1)        # теж у 1D (-1 = автоматично)

print(m)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

# Транспонування матриці
print(m.T)               # рядки і стовпці міняються місцями
print(m.T.shape)         # (4, 3)
```

Арифметичні операції. Векторизація.

Головна перевага NumPy — операції над масивами виконуються поелементно без явних циклів. Це називається векторизацією і дозволяє писати компактний та швидкий код.

```
import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([10, 20, 30, 40])

# ── Поелементні операції ──
print(a + b)      # [11 22 33 44]
print(a * b)      # [10 40 90 160]
print(b / a)      # [10. 10. 10. 10.]
print(a ** 2)     # [ 1  4  9 16]

# ── Broadcasting: операція масиву зі скаляром ──
```

```
# (scalar застосовується до КОЖНОГО елемента автоматично)
print(a * 3)      # [ 3  6  9 12]
print(a + 100)    # [101 102 103 104]
print(a > 2)      # [False False  True  True]

# ── Математичні функції (поелементні) ──
x = np.linspace(0, np.pi, 5)
print(np.sin(x))  # [0.    0.707 1.    0.707 0.   ]  (приблизно)
print(np.cos(x))
print(np.exp(x))  # e^x для кожного елемента
print(np.sqrt(np.array([1, 4, 9, 16])))  # [1. 2. 3. 4.]
print(np.log(np.array([1, np.e, np.e**2])))  # [0. 1. 2.]
print(np.abs(np.array([-3, 1, -5, 2])))  # [3 1 5 2]

# ── Агрегуючі функції ──
data = np.array([4, 7, 2, 9, 1, 5, 8, 3, 6])
print(data.sum())      # 45
print(data.mean())     # 5.0  — середнє
print(data.std())      # стандартне відхилення
print(data.min())      # 1
print(data.max())      # 9
print(data.argmin())   # 4  — ІНДЕКС мінімального
print(data.argmax())   # 3  — ІНДЕКС максимального
print(np.median(data)) # 5.0  — медіана

# ── Агрегація по осях матриці ──
m = np.array([[1, 2, 3],
              [4, 5, 6]])
print(m.sum(axis=0))  # [5 7 9]   — сума по стовпцях
print(m.sum(axis=1))  # [6 15]    — сума по рядках
print(m.mean(axis=1)) # [2. 5.]   — середнє по рядках
```

Лінійна алгебра (`np.linalg`). NumPy має вбудований підмодуль `linalg` (linear algebra) для розв'язання задач лінійної алгебри, що вивчаються у курсах вищої математики та фізики.

```
import numpy as np

# ── Множення матриць ──
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

print(A @ B)        # матричне множення (оператор @)
# [[19 22]
#  [43 50]]

print(np.dot(A, B)) # те саме через np.dot()

# ── Транспонування ──
print(A.T)
# [[1 3]
#  [2 4]]

# ── Визначник матриці ──
```

```
print(np.linalg.det(A))   # -2.0

# ── Обернена матриця ──
A_inv = np.linalg.inv(A)
print(A_inv)
# [[-2.   1. ]
#  [ 1.5 -0.5]]

# Перевірка: A * A^(-1) = I (одинична матриця)
print(np.round(A @ A_inv, 10))
# [[1. 0.]
#  [0. 1.]]

# ── Розв'язання системи лінійних рівнянь ──
# 2x + 3y = 13
# x  - 2y = -4
A_sys = np.array([[2, 3],
                  [1, -2]])
b_vec = np.array([13, -4])

x_sol = np.linalg.solve(A_sys, b_vec)
print(x_sol)          # [2. 3.]  → x=2, y=3

# ── Власні значення та власні вектори ──
M = np.array([[4, 1], [2, 3]])
eigenvalues, eigenvectors = np.linalg.eig(M)
print('Власні значення:', eigenvalues)   # [5. 2.]
print('Власні вектори:\n', eigenvectors)

# ── Норма вектора ──
v = np.array([3, 4])
print(np.linalg.norm(v))  # 5.0  — довжина вектора
```

Статистика та обробка даних

```
import numpy as np

# Змоделюємо дані: оцінки 30 студентів
np.random.seed(42)   # фіксуємо генератор для відтворюваності
grades = np.random.randint(50, 101, size=30)
print(grades)

# ── Описова статистика ──
print(f'Середнє:     {grades.mean():.2f}')
print(f'Медіана:     {np.median(grades):.1f}')
print(f'Std:         {grades.std():.2f}')
print(f'Дисперсія:   {grades.var():.2f}')
print(f'Мін/Макс:    {grades.min()} / {grades.max()}')
print(f'Перцентиль 25: {np.percentile(grades, 25)}')
print(f'Перцентиль 75: {np.percentile(grades, 75)}')
```

```
# ── Сортування ──
sorted_grades = np.sort(grades)        # новий відсортований масив
grades.sort()                          # сортує масив на місці
print(np.argsort(grades))              # індекси відсортованих елементів

# ── Підрахунок унікальних значень ──
categories = np.array(['A','B','A','C','B','A','C','C'])
values, counts = np.unique(categories, return_counts=True)
for v, c in zip(values, counts):
    print(f'{v}: {c} разів')

# ── Коефіцієнт кореляції ──
height = np.array([165, 170, 175, 180, 185])
weight = np.array( [55,  65,  72,  80,  88])
corr = np.corrcoef(height, weight)
print(f'Кореляція зріст-вага: {corr[0,1]:.4f}')

# ── Апроксимація поліномом (метод найменших квадратів) ──
x = np.array([1, 2, 3, 4, 5])
y = np.array([2.1, 3.9, 6.2, 7.8, 10.1])
coeffs = np.polyfit(x, y, deg=1)   # лінійна апроксимація
print(f'y ≈ {coeffs[0]:.2f}x + {coeffs[1]:.2f}')

poly_func = np.poly1d(coeffs)      # об'єкт полінома
print(poly_func(3.5))              # значення у при x=3.5
```

Практичні інженерні та фізичні задачі з NumPy.
Розглянемо кілька типових задач, що зустрічаються у курсах фізики та інженерних дисциплін.

Приклад 1: Рух тіла, кинутого під кутом

```
import numpy as np

# Параметри
v0    = 20.0          # початкова швидкість, м/с
angle = 45.0          # кут кидання, градуси
g     = 9.81          # прискорення вільного падіння

# Переводимо кут у радіани
theta = np.radians(angle)

# Проекції початкової швидкості
vx = v0 * np.cos(theta)
vy = v0 * np.sin(theta)

# Час польоту
T = 2 * vy / g

# Масив моментів часу: 200 точок від 0 до T
t = np.linspace(0, T, 200)
```

```
# Координати у кожен момент часу (векторизовано!)
x = vx * t
y = vy * t - 0.5 * g * t**2

# Дальність і максимальна висота
print(f'Дальність:       {x[-1]:.2f} м')
print(f'Макс. висота:    {y.max():.2f} м')
print(f'Час польоту:     {T:.2f} с')
# x та y — готові масиви для побудови графіку траєкторії
```

Приклад 2: Розв'язання системи рівнянь (закон Кірхгофа)

```
import numpy as np

# Система рівнянь для трьох вузлів електричного кола:
# R1*I1 + R2*(I1-I2) = U1
# R3*I2 + R2*(I2-I1) = 0
# R4*(I1+I2) = U2

# Спрощена модель: 3 рівняння, 3 невідомих
# 5*x + 2*y - z  = 10
# 2*x + 8*y + 3*z = 20
#  x + y + 10*z = 15

R = np.array([[5,  2, -1],
              [2,  8,  3],
              [1,  1, 10]])

U = np.array([10, 20, 15])

I = np.linalg.solve(R, U)
print('Струми (A):')
for i, val in enumerate(I, 1):
    print(f'  I{i} = {val:.4f} A')

# Перевірка: R @ I повинно дорівнювати U
print('Перевірка:', np.allclose(R @ I, U))  # True
```

Частина ІІ. Бібліотека Matplotlib.

`Matplotlib` — найпоширеніша бібліотека для побудови графіків у Python. Основний модуль — `pyplot` (зазвичай імпортується як plt).

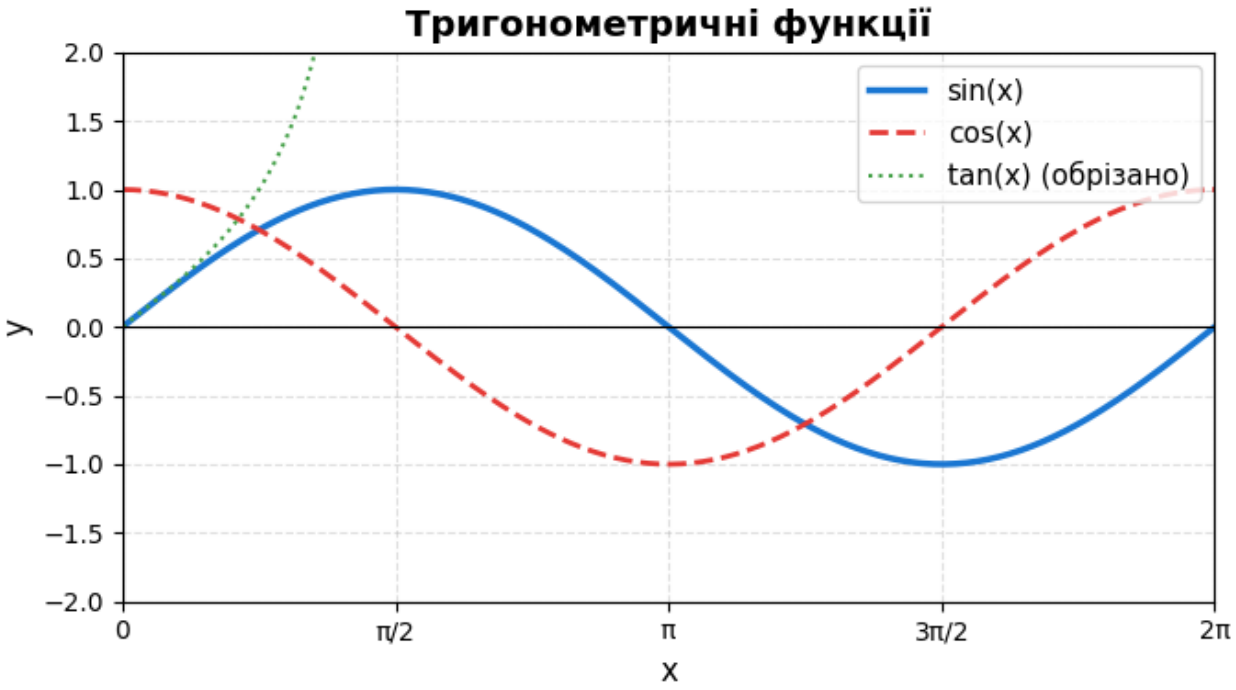
Два рівні роботи з Matplotlib:
pyplot API (простий): `plt.plot()`, `plt.show()` — зручний для швидкої побудови
Object-oriented API (гнучкий): `fig, ax = plt.subplots()` — рекомендований для складних графіків

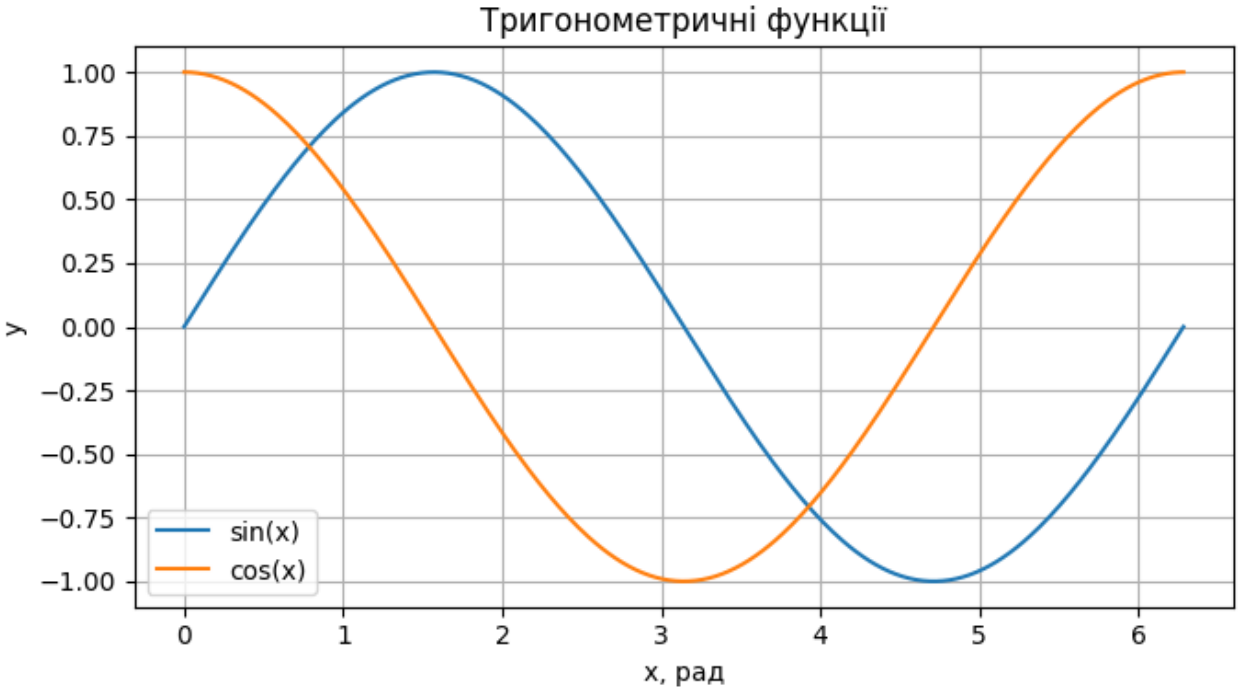
Синтаксис	Опис / Результат
`plt.figure()`	Створює нове полотно (Figure). Параметр `figsize=(w, h)` задає розміри у дюймах.
`plt.subplots()`	Створює Figure та один або масив Axes (підграфіків).
`ax.plot(x, y)`	Лінійний графік. Параметри: color, linewidth, linestyle, label.
`ax.set_title(s)`	Заголовок графіка.
`ax.set_xlabel(s)`	Підпис осі X.
`ax.set_ylabel(s)`	Підпис осі Y.
`ax.legend()`	Відображає легенду (для кривих із label=).
`ax.grid(True)`	Відображає сітку.
`ax.set_xlim(a,b)`	Встановлює межі осі X.
`ax.set_ylim(a,b)`	Встановлює межі осі Y.
`plt.savefig(path)`	Зберігає рисунок у файл. Формат визначається розширенням.
`plt.show()`	Відображає графік у вікні (не потрібний при savefig у скриптах).

Лінійні графіки. Форматування

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 300)

# ── Базовий графік ──
fig, ax = plt.subplots(figsize=(7, 4))

ax.plot(x, np.sin(x), label='sin(x)')
ax.plot(x, np.cos(x), label='cos(x)')

ax.set_title('Тригонометричні функції')
ax.set_xlabel('x, рад')
ax.set_ylabel('y')
ax.legend()
ax.grid(True)
plt.tight_layout()
plt.savefig('trig.png', dpi=150)
plt.show()

# ── Детальне форматування кривих ──
```

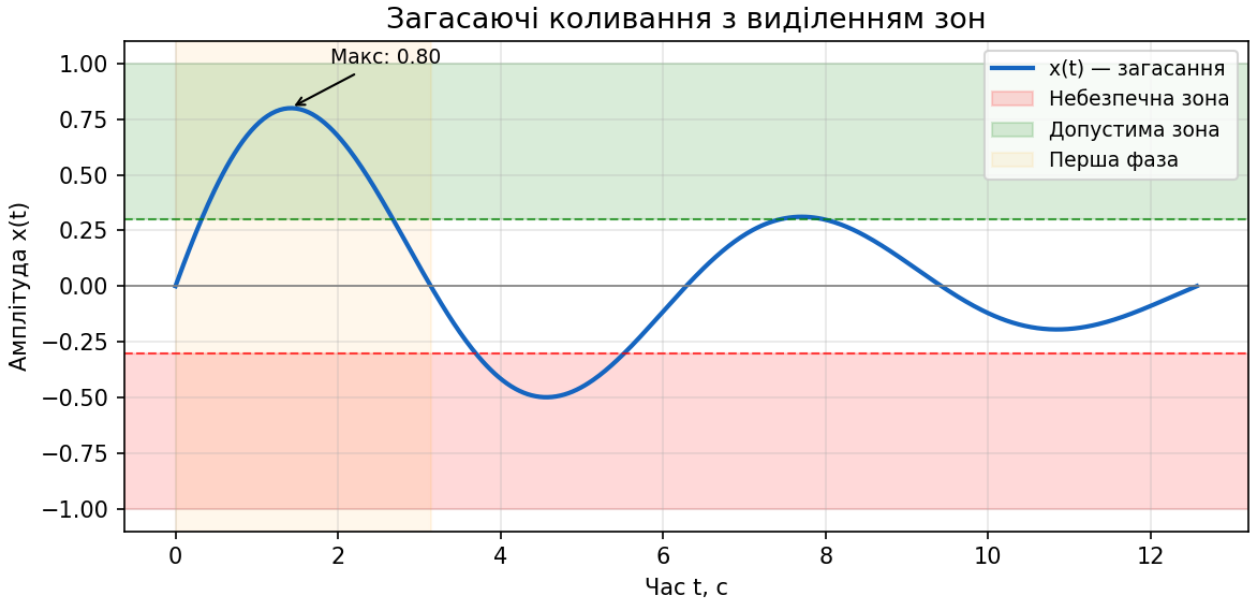
```python
fig, ax = plt.subplots(figsize=(7, 4))

ax.plot(x, np.sin(x),
        color='#1976D2',        # колір (hex)
        linewidth=2.5,          # товщина лінії
        linestyle='-',          # суцільна лінія
        label='sin(x)')

ax.plot(x, np.cos(x),
        color='#E53935',
        linewidth=2,
        linestyle='--',         # пунктирна
        label='cos(x)')

ax.plot(x, np.tan(np.clip(x, -1.4, 1.4)),
        color='#43A047',
        linewidth=1.5,
        linestyle=':',          # крапкова
        label='tan(x) (обрізано)')

# Форматування осей і підписів
ax.set_xlim(0, 2*np.pi)
ax.set_ylim(-2, 2)
ax.set_xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi])
ax.set_xticklabels(['0', 'π/2', 'π', '3π/2', '2π'])
ax.set_title('Тригонометричні функції', fontsize=14, fontweight='bold')
ax.set_xlabel('x', fontsize=12)
ax.set_ylabel('y', fontsize=12)
ax.legend(fontsize=11, loc='upper right')
ax.grid(True, alpha=0.4, linestyle='--')
ax.axhline(0, color='black', linewidth=0.8)  # вісь X
ax.axvline(0, color='black', linewidth=0.8)  # вісь Y

plt.tight_layout()
plt.savefig('trig_styled.png', dpi=150, bbox_inches='tight')
plt.show()
```


Тригонометричні функції
sin(x)
cos(x)
tan(x) (обрізано)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
0
π/2
π
3π/2
2π
x


Тригонометричні функції
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
sin(x)
cos(x)
0
1
2
3
4
5
6
x, рад

Виділення зон та фрагментів графіка. Matplotlib дозволяє підсвічувати окремі ділянки графіка — це дуже корисно для виділення критичних зон, допусків, фаз руху тощо.

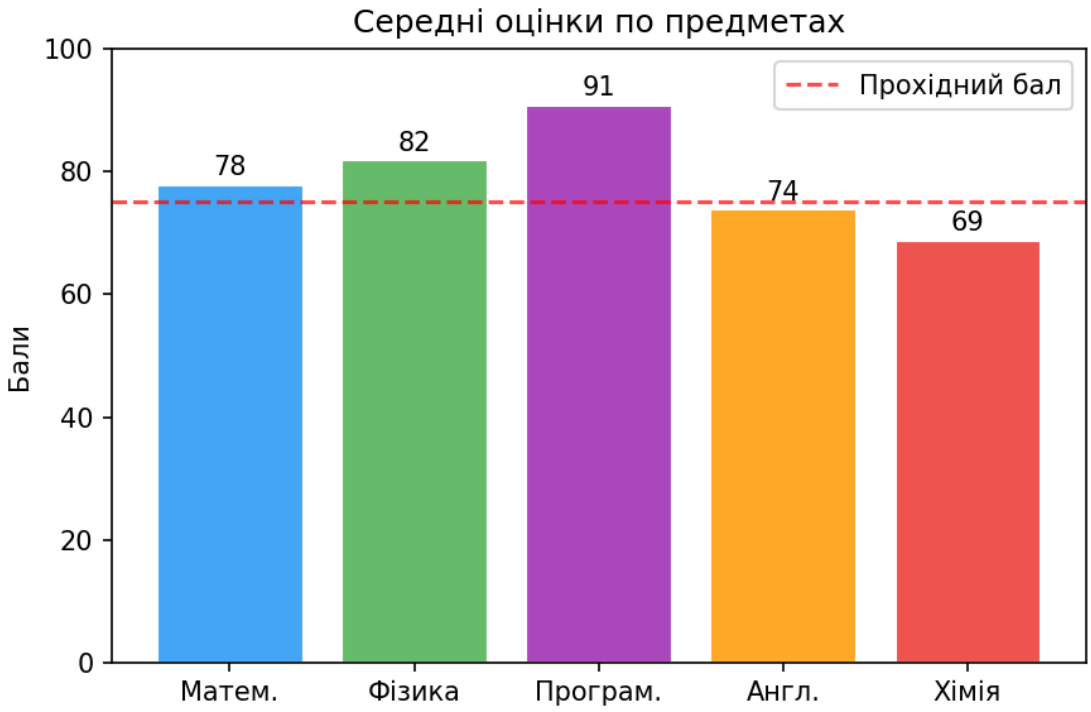
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 4 * np.pi, 500)
y = np.sin(x) * np.exp(-x * 0.15)   # загасаючі коливання

fig, ax = plt.subplots(figsize=(8, 4))

# Основна крива
ax.plot(x, y, color='#1565C0', linewidth=2, label='x(t) — загасання')

# ── Виділення небезпечної зони (axhspan) ──
# Горизонтальний span: виділяє зону між y=-0.3 і y=-1
ax.axhspan(-1.0, -0.3, alpha=0.15, color='red',
           label='Небезпечна зона')
ax.axhspan(0.3,   1.0, alpha=0.15, color='green',
           label='Допустима зона')

# ── Виділення фази часу (axvspan) ──
# Вертикальний span: виділяє часовий проміжок
ax.axvspan(0, np.pi, alpha=0.08, color='orange',
           label='Перша фаза')

# ── Горизонтальна лінія допуску ──
ax.axhline(0, color='gray', linewidth=0.8, linestyle='-')
ax.axhline( 0.3, color='green', linewidth=1, linestyle='--', alpha=0.7)
ax.axhline(-0.3, color='red',   linewidth=1, linestyle='--', alpha=0.7)

# ── Позначення конкретної точки ──
i_max = np.argmax(y)
ax.annotate(f'Макс: {y[i_max]:.2f}',
            xy=(x[i_max], y[i_max]),
            xytext=(x[i_max]+0.5, y[i_max]+0.2),
            arrowprops=dict(arrowstyle='->', color='black'),
            fontsize=9)

ax.set_title('Загасаючі коливання з виділенням зон', fontsize=13)
ax.set_xlabel('Час t, с')
ax.set_ylabel('Амплітуда x(t)')
ax.legend(fontsize=9, loc='upper right')
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('damping_zones.png', dpi=150)
plt.show()
```

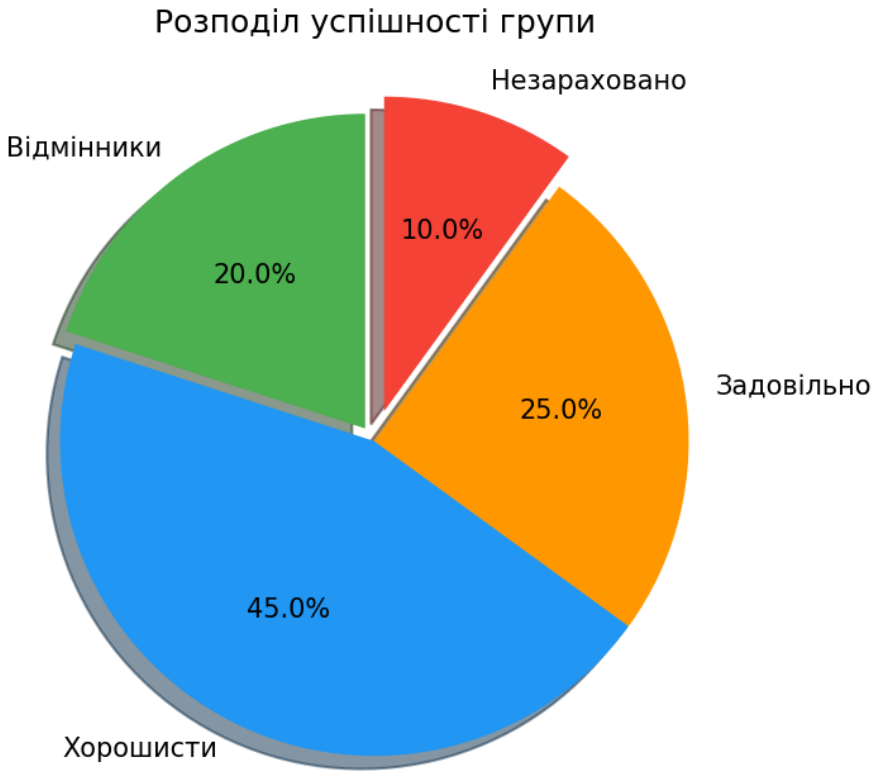


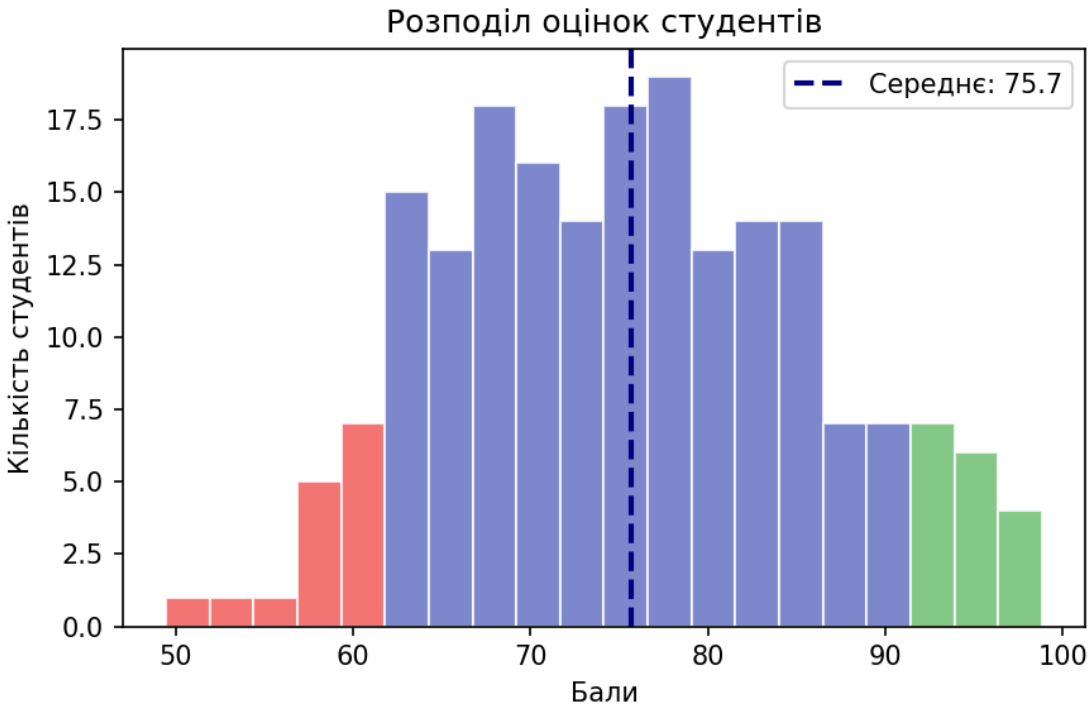
Різні типи графіків

```python
import numpy as np
import matplotlib.pyplot as plt

# — Стовпчаста діаграма (bar) —
subjects = ['Матем.', 'Фізика', 'Програм.', 'Англ.', 'Хімія']
avg_scores = [78, 82, 91, 74, 69]
colors = ['#42A5F5','#66BB6A','#AB47BC','#FFA726','#EF5350']

fig, ax = plt.subplots(figsize=(6, 4))
bars = ax.bar(subjects, avg_scores, color=colors, edgecolor='white',
              linewidth=1.5)

# Підписи значень над стовпцями
for bar, val in zip(bars, avg_scores):
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.5,
            str(val), ha='center', va='bottom', fontsize=10)

ax.set_ylim(0, 100)
ax.set_title('Середні оцінки по предметах')
ax.set_ylabel('Бали')
ax.axhline(75, color='red', linestyle='--', alpha=0.7, label='Прохідний бал')
ax.legend()
plt.tight_layout()
plt.savefig('bar_chart.png', dpi=150)
plt.show()

# — Кругова діаграма (pie) —
categories = ['Відмінники', 'Хорошисти', 'Задовільно', 'Незараховано']
sizes = [20, 45, 25, 10]
```

```
explode = (0.05, 0, 0, 0.1)   # виділити скибки
pie_colors = ['#4CAF50','#2196F3','#FF9800','#F44336']

fig, ax = plt.subplots(figsize=(5, 5))
ax.pie(sizes, labels=categories, explode=explode,
       colors=pie_colors, autopct='%1.1f%%',
       startangle=90, shadow=True)
ax.set_title('Розподіл успішності групи')
plt.tight_layout()
plt.savefig('pie_chart.png', dpi=150)
plt.show()

# ── Гістограма (hist) ──
np.random.seed(0)
data = np.random.normal(75, 10, 200)  # нормальний розподіл

fig, ax = plt.subplots(figsize=(6, 4))
n, bins, patches = ax.hist(data, bins=20, color='#5C6BC0',
                           edgecolor='white', alpha=0.8)

# Виділити кольором різні діапазони
for patch, left in zip(patches, bins[:-1]):
    if left < 60:
        patch.set_facecolor('#EF5350')
    elif left >= 90:
        patch.set_facecolor('#66BB6A')

ax.axvline(data.mean(), color='navy', linewidth=2,
           linestyle='--', label=f'Середнє: {data.mean():.1f}')
ax.set_title('Розподіл оцінок студентів')
ax.set_xlabel('Бали')
ax.set_ylabel('Кількість студентів')
ax.legend()
plt.tight_layout()
plt.savefig('histogram.png', dpi=150)
plt.show()
```


Середні оцінки по предметах
Прохідний бал
Бали
100
80
60
40
20
0
78
82
91
74
69
Матем.
Фізика
Програм.
Англ.
Хімія

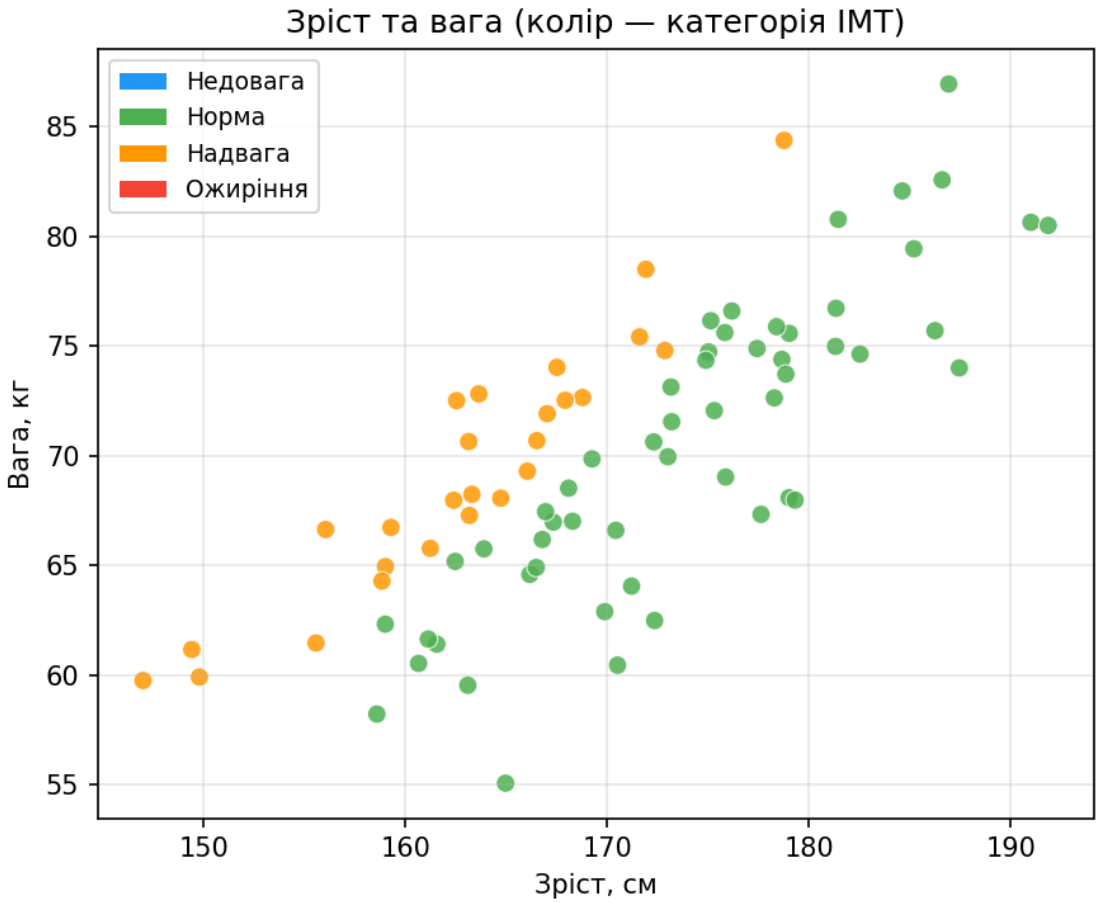

Розподіл успішності групи
Незараховано
Відмінники
10.0%
20.0%
25.0%
Задовільно
45.0%
Хорошисти

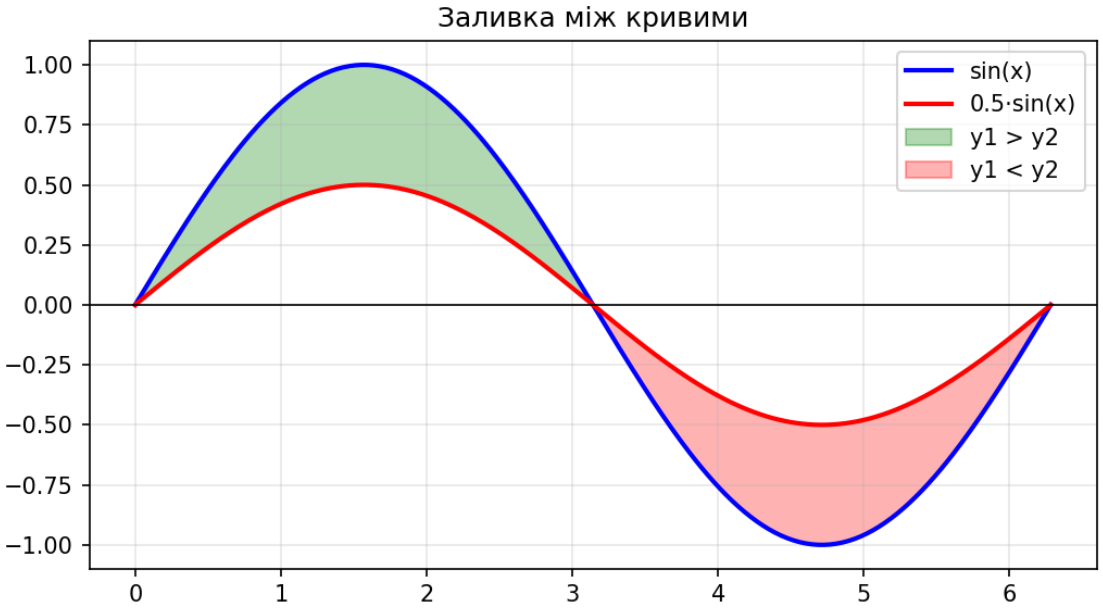


Діаграма розсіювання. Заливка між кривими

```python
import numpy as np
import matplotlib.pyplot as plt

# — Scatter (точкова діаграма) —
np.random.seed(1)
n = 80
height = np.random.normal(170, 10, n)
weight = height * 0.45 - 7 + np.random.normal(0, 5, n)
bmi    = weight / (height/100)**2

# Колір точки залежить від ІМТ
colors = np.where(bmi < 18.5, '#2196F3',
         np.where(bmi < 25,   '#4CAF50',
         np.where(bmi < 30,   '#FF9800', '#F44336')))

fig, ax = plt.subplots(figsize=(6, 5))
scatter = ax.scatter(height, weight, c=colors, s=50,
                     edgecolors='white', linewidths=0.5, alpha=0.85)

# Ручна легенда
from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor='#2196F3', label='Недовага'),
    Patch(facecolor='#4CAF50', label='Норма'),
    Patch(facecolor='#FF9800', label='Надвага'),
    Patch(facecolor='#F44336', label='Ожиріння'),
]
ax.legend(handles=legend_elements, fontsize=9)
```

```
ax.set_title('Зріст та вага (колір — категорія ІМТ)')
ax.set_xlabel('Зріст, см')
ax.set_ylabel('Вага, кг')
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('scatter_bmi.png', dpi=150)
plt.show()

# ── fill_between — заливка між кривими ──
x = np.linspace(0, 2*np.pi, 300)
y1 = np.sin(x)
y2 = np.sin(x) * 0.5

fig, ax = plt.subplots(figsize=(7, 4))
ax.plot(x, y1, 'b-', linewidth=2, label='sin(x)')
ax.plot(x, y2, 'r-', linewidth=2, label='0.5·sin(x)')

# Заливка між кривими (де y1 > y2 — зелено, де y1 < y2 — червоно)
ax.fill_between(x, y1, y2,
                where=(y1 >= y2), alpha=0.3, color='green',
                label='y1 > y2')
ax.fill_between(x, y1, y2,
                where=(y1 < y2),  alpha=0.3, color='red',
                label='y1 < y2')

ax.axhline(0, color='black', linewidth=0.8)
ax.set_title('Заливка між кривими')
ax.legend()
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('fill_between.png', dpi=150)
plt.show()
```

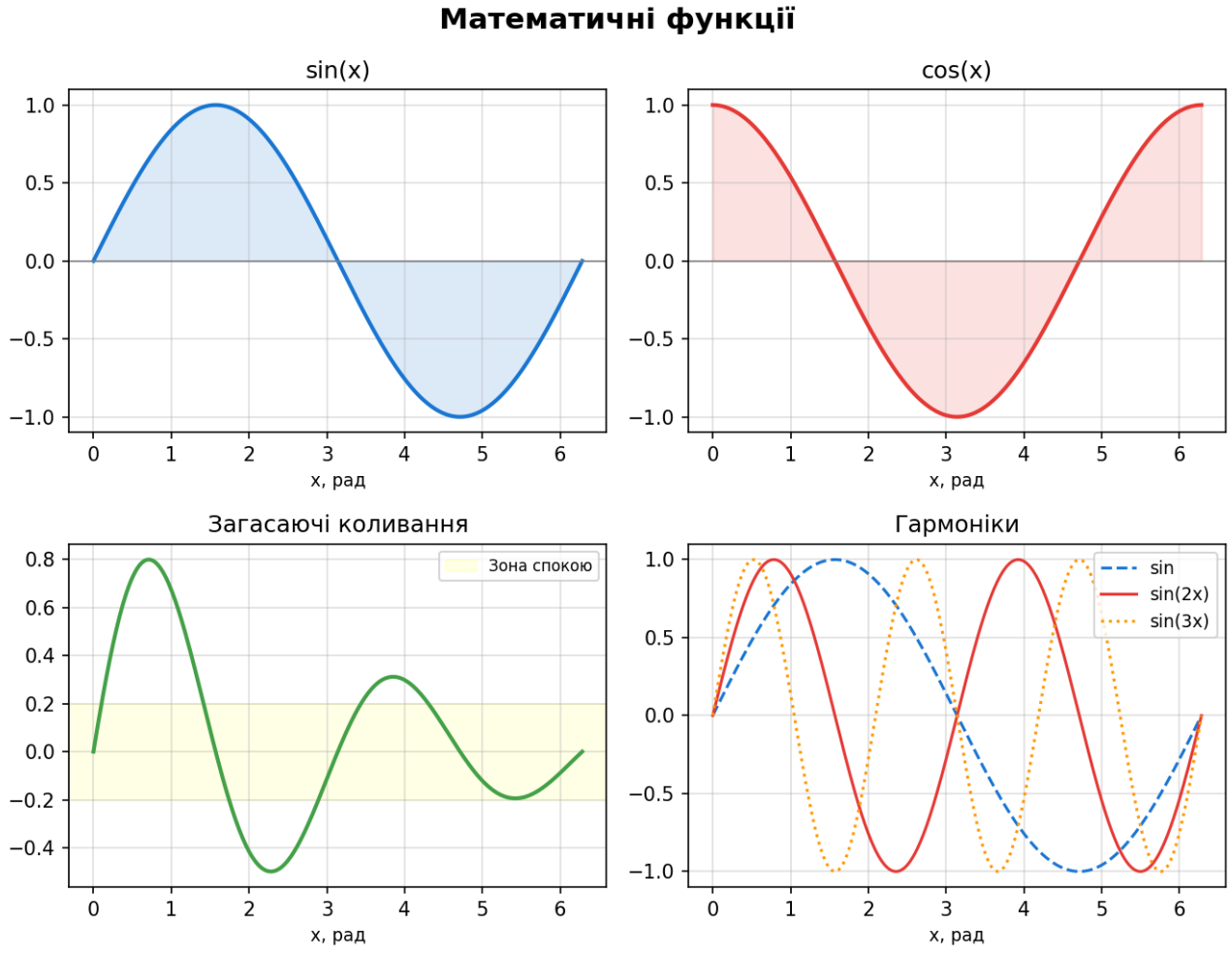

Зріст та вага (колір — категорія ІМТ)
Недовага
Норма
Надвага
Ожиріння
Вага, кг
Зріст, см
55
60
65
70
75
80
85
150
160
170
180
190

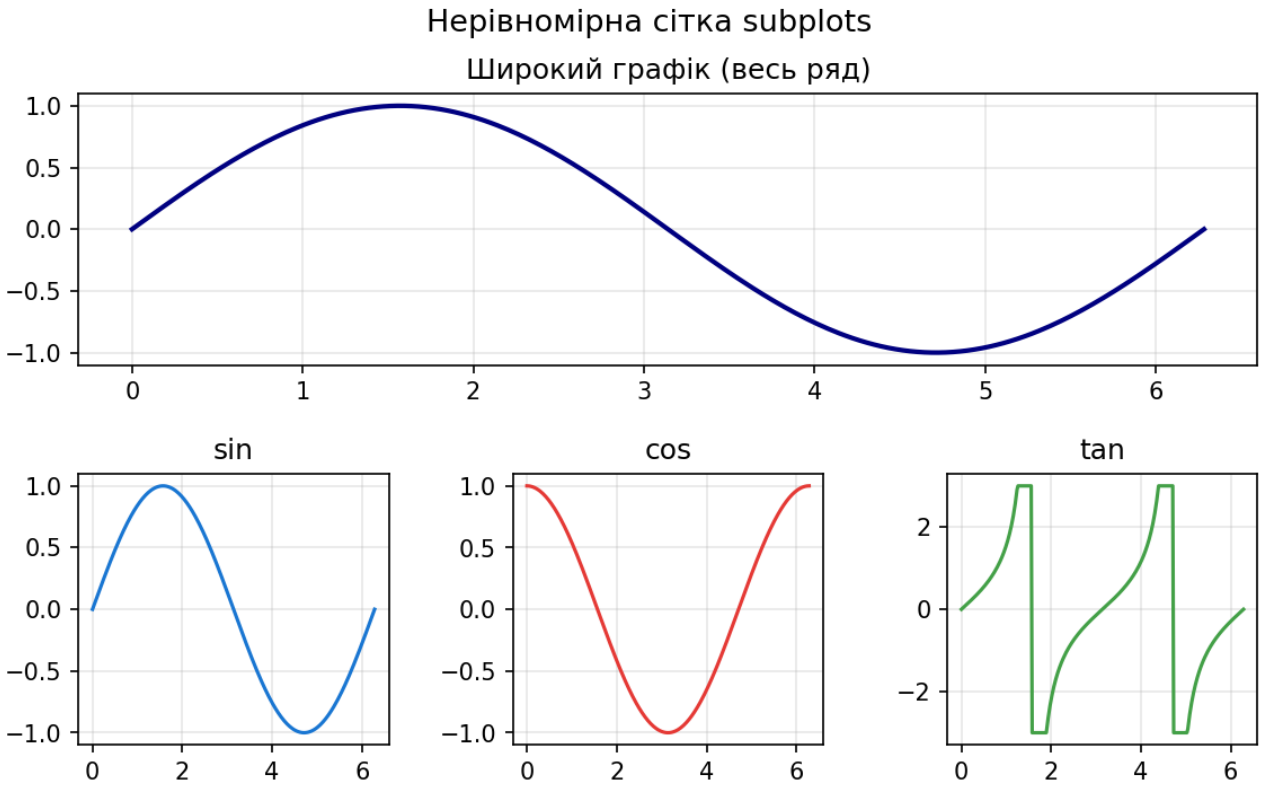

Заливка між кривими
sin(x)
0.5·sin(x)
y1 > y2
y1 < y2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6

Subplots: кілька графіків на одному полотні

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 300)

# ── 2×2 сітка підграфіків ──
fig, axes = plt.subplots(2, 2, figsize=(9, 7))
fig.suptitle('Математичні функції', fontsize=15, fontweight='bold')

# Підграфік [0,0] — sin(x)
axes[0, 0].plot(x, np.sin(x), color='#1976D2', linewidth=2)
axes[0, 0].fill_between(x, np.sin(x), 0, alpha=0.15, color='#1976D2')
axes[0, 0].set_title('sin(x)')
axes[0, 0].grid(True, alpha=0.4)
axes[0, 0].axhline(0, color='gray', linewidth=0.8)

# Підграфік [0,1] — cos(x)
axes[0, 1].plot(x, np.cos(x), color='#E53935', linewidth=2)
axes[0, 1].fill_between(x, np.cos(x), 0, alpha=0.15, color='#E53935')
axes[0, 1].set_title('cos(x)')
axes[0, 1].grid(True, alpha=0.4)
axes[0, 1].axhline(0, color='gray', linewidth=0.8)

# Підграфік [1,0] — exp(-x)*sin(x) загасаючі коливання
y_damp = np.exp(-x*0.3) * np.sin(2*x)
axes[1, 0].plot(x, y_damp, color='#43A047', linewidth=2)
axes[1, 0].axhspan(-0.2, 0.2, alpha=0.1, color='yellow',
                   label='Зона спокою')
axes[1, 0].set_title('Загасаючі коливання')
axes[1, 0].legend(fontsize=8)
axes[1, 0].grid(True, alpha=0.4)
# Підграфік [1,1] — кілька функцій разом
axes[1, 1].plot(x, np.sin(x),   '--', color='#1976D2', label='sin')
axes[1, 1].plot(x, np.sin(2*x), '-',  color='#E53935', label='sin(2x)')
axes[1, 1].plot(x, np.sin(3*x), ':',  color='#FF9800', label='sin(3x)')
axes[1, 1].set_title('Гармоніки')
axes[1, 1].legend(fontsize=9)
axes[1, 1].grid(True, alpha=0.4)

# Спільний підпис осей
for ax in axes.flat:
    ax.set_xlabel('x, рад', fontsize=9)

plt.tight_layout()
plt.savefig('subplots_2x2.png', dpi=150, bbox_inches='tight')
plt.show()

# ── Нерівномірна сітка через gridspec ──
fig = plt.figure(figsize=(9, 5))
gs  = fig.add_gridspec(2, 3, hspace=0.4, wspace=0.4)
```

```
ax_wide = fig.add_subplot(gs[0, :])   # верхній рядок — весь ряд
ax_bl   = fig.add_subplot(gs[1, 0])
ax_bm   = fig.add_subplot(gs[1, 1])
ax_br   = fig.add_subplot(gs[1, 2])

ax_wide.plot(x, np.sin(x), color='navy', linewidth=2)
ax_wide.set_title('Широкий графік (весь ряд)')
ax_wide.grid(True, alpha=0.3)

for ax, func, clr, lbl in [
    (ax_bl, np.sin, '#1976D2', 'sin'),
    (ax_bm, np.cos, '#E53935', 'cos'),
    (ax_br, np.tan, '#43A047', 'tan'),
]:
    ax.plot(x, np.clip(func(x),-3,3), color=clr, linewidth=1.5)
    ax.set_title(lbl)
    ax.grid(True, alpha=0.3)

fig.suptitle('Нерівномірна сітка subplots', fontsize=13)
plt.savefig('gridspec.png', dpi=150, bbox_inches='tight')
plt.show()
```





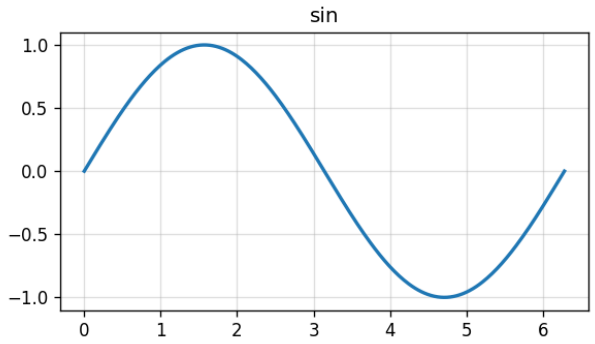
Збереження графіків у файл. Функція `plt.savefig()` зберігає поточний графік у файл. Формат визначається розширенням імені файлу.

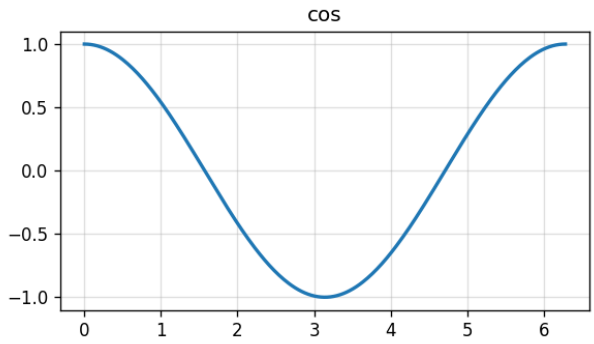
Синтаксис	Опис / Результат
`.png`	PNG — растровий, з прозорістю. Найбільш поширений для звітів.
`.pdf`	PDF — векторний, масштабується без втрат. Ідеальний для публікацій.
`.svg`	SVG — векторний XML. Можна редагувати у Inkscape/Illustrator.
`.eps`	EPS — векторний, для LaTeX та наукових журналів.
`.jpg`	JPEG — стиснений, без прозорості. Менший розмір, але артефакти.

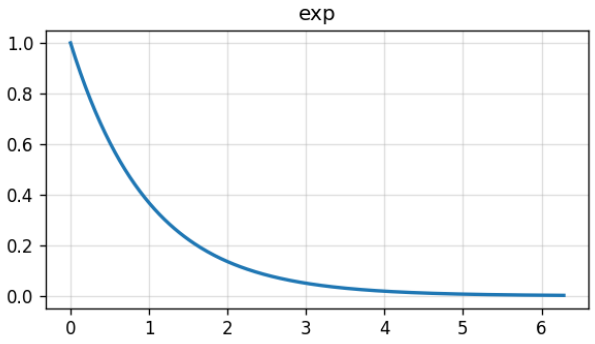
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2*np.pi, 300)
fig, ax = plt.subplots(figsize=(7, 4))
ax.plot(x, np.sin(x), linewidth=2)
ax.set_title('Приклад збереження')

# Основні параметри savefig:
# dpi    — роздільна здатність (72 — екран, 150–300 — друк)
# bbox_inches='tight' — обрізати зайві поля
# transparent=True    — прозорий фон (для PNG)
```

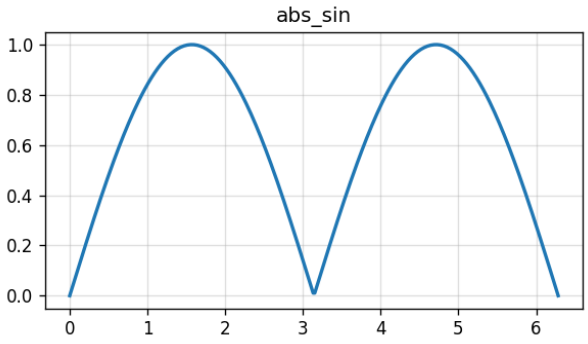
```
# facecolor          — колір фону рисунку

plt.savefig('graph.png',  dpi=150, bbox_inches='tight')
plt.savefig('graph.pdf',  bbox_inches='tight')
plt.savefig('graph.svg',  bbox_inches='tight')
plt.savefig('graph_transparent.png', dpi=150,
            transparent=True, bbox_inches='tight')

# Збереження у визначену папку
from pathlib import Path
output_dir = Path('lab7_output')
output_dir.mkdir(exist_ok=True)

plt.savefig(output_dir / 'sine_wave.png', dpi=200)
print('Графік збережено!')

plt.show()

# Збереження кількох графіків у цикл
functions = {'sin': np.sin, 'cos': np.cos,
             'exp': lambda x: np.exp(-x),
             'abs_sin': lambda x: np.abs(np.sin(x))}

for name, func in functions.items():
    fig, ax = plt.subplots(figsize=(5, 3))
    ax.plot(x, func(x), linewidth=2)
    ax.set_title(name)
    ax.grid(True, alpha=0.4)
    plt.tight_layout()
    plt.savefig(output_dir / f'{name}.png', dpi=120)
    plt.close()   # ВАЖЛИВО: закривати після збереження у циклі!
    print(f'Збережено: {name}.png')
```

Приклад збереження
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6


sin
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
6


cos
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
6


exp
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6


abs_sin
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6

Завдання на лабораторну роботу;

Перед виконанням встановіть бібліотеки:

```
pip install numpy matplotlib scipy
```

Усі графіки зберігайте у папці `lab7_output/` у форматах PNG та PDF.

7.1. Масиви NumPy та базові операції. Виконайте такі дії з масивами:

- створіть масив `a` з 20 рівномірно розміщених чисел від 0 до 10 (`np.linspace`) та масив `b` з 20 цілих випадкових чисел від 1 до 50;
- виведіть: суму, добуток, різницю масивів; масив `a` у квадраті; масив `b`, де всі непарні елементи замінено на 0 (булеве індексування);
- знайдіть індекси елементів масиву `b`, що перевищують 30, виведіть самі ці елементи;
- перетворіть `b` у матрицю 4×5, виведіть суму по рядках і стовпцях; транспонуйте матрицю.

7.2. Тригонометричні функції та їх графіки. Побудуйте на одному полотні (subplots 2×1) два графіки:

- верхній: `sin(x)`, `cos(x)` і `sin(x)+cos(x)` на проміжку `[0, 4π]` з різними кольорами, стилями ліній і легендою;
- нижній: `sin²(x)` і `cos²(x)`, а між ними — заливка `fill_between`; поясніть словами (у `title` або `annotation`), чому `sin²(x)+cos²(x)=1`;
- Виділіть осьовими лініями `axhline/axvline` нулі, позначте максимуми кривих маркерами

Збережіть у trig_lab.png та trig_lab.pdf

7.3. Статистика і гістограма. Згенеруйте масив із 500 значень нормального розподілу N(70, 12) — імітація оцінок великого курсу. Побудуйте на одному полотні (1×2):

- зліва: гістограму з 25 стовпцями; виділіть червоним стовпці до 60 балів, зеленим — від 90; нанесіть вертикальні лінії середнього та медіани;
- справа: кругову діаграму розподілу по категоріях (< 60, 60–74, 75–89, ≥ 90).

Виведіть у консоль описову статистику: середнє, медіана, std, min, max, перцентилі 10 і 90.

7.4. Фізика: рух тіла, кинутого під кутом. На основі прикладу з теоретичної частини (розділ 3.6) побудуйте інтерактивне дослідження:

- для кутів 30°, 45°, 60° і 75° за однакової початкової швидкості v_0 = 25 м/с обчисліть траєкторії (масиви x та y через NumPy);
- побудуйте всі траєкторії на одному графіку різними кольорами, у легенді вкажіть кут і дальність польоту;
- виділіть зону `axhspan` від 0 до 5 м — «зона перешкоди»; побудуйте аннотацію для максимальної висоти кожної траєкторії;
- на окремому підграфіку (subplots 1×2) покажіть залежність дальності польоту від кута (від 0° до 90° через 1°) та залежність максимальної висоти від кута.

Збережіть як projectile_motion.png і projectile_motion.pdf.

7.5. Лінійна алгебра: системи рівнянь та матриці. Виконайте такі задачі з np.linalg:

- задайте матрицю A 3×3 з довільними ненульовими числами. Обчисліть: det(A), A^{-1},

$A \cdot A^{-1}$ (перевірка = одинична). Виведіть результати з поясненнями;

- розв'яжіть систему трьох лінійних рівнянь (придумайте власну або використайте з курсу вищої математики). Перевірте правильність через підстановку;
- знайдіть власні значення та власні вектори матриці. Побудуйте на графіку (стрілки quiver або plot) початкові та трансформовані вектори — наочно покажіть, як матриця розтягує/повертає вектори.

Контрольні запитання

1. Що таке масив `ndarray` у NumPy? Чим він відрізняється від звичайного списку Python за структурою зберігання та швидкістю обчислень?
2. Що таке векторизація у NumPy? Поясніть на прикладі, чому 'a * 3' для NumPy-масиву швидше, ніж цикл for.
3. Що означають атрибути `shape`, `ndim`, `size` та `dtype` масиву? Що поверне вираз `np.zeros((3,4)).shape`?
4. Чим відрізняються `np.arange()` та `np.linspace()`? Коли доцільно використовувати кожну з них?
5. Поясніть принцип булевого індексування. Що поверне вираз `a[a > 5]` для масиву `a = np.array([3, 7, 1, 9, 4])`?
6. Що таке `broadcasting` у NumPy? Як виконується операція між масивом форми (3,4) і скаляром?
7. Для чого використовується `np.linalg.solve()`? Запишіть, яку задачу розв'язує цей виклик: `np.linalg.solve(A, b)`.
8. Що таке `Figure` та `Axes` у Matplotlib? Поясніть різницю між `plt.plot()` та `ax.plot()`.
9. Для чого використовуються функції `axhspan()` та `axvspan()`? Наведіть практичний приклад їхнього застосування.
10. Назвіть чотири формати збереження графіків у Matplotlib. Які переваги PNG перед PDF і навпаки?

Лабораторна робота №8.

Основи об'єктно-орієнтованого програмування в Python

Мета роботи: Засвоїти фундаментальні концепції об'єктно-орієнтованого програмування: навчитися визначати класи, створювати об'єкти та керувати їхніми атрибутами й методами; зрозуміти принципи інкапсуляції через механізми public, protected та private. Ознайомитися з різними видами методів (екземпляра, класовими, статичними), навчитися використовувати декоратор @property для контрольованого доступу до атрибутів; опанувати механізм успадкування (одиночне й множинне), порядок вирішення методів (MRO) та перевизначення методів. Вивчити абстрактні класи, спеціальні (dunder/magic) методи, протоколи та найкращі практики проектування класів у Python; навчитися застосовувати ООП для побудови реальних програм зі складною ієрархією об'єктів.

Теоретичні відомості:

Об'єктно-орієнтоване програмування (ООП) — парадигма, де програма моделюється як набір об'єктів, що взаємодіють між собою. Кожен об'єкт поєднує дані (атрибути) і поведінку (методи).

Чотири стовпи ООП:

- Інкапсуляція — приховання внутрішньої реалізації, доступ через інтерфейс
- Успадкування — клас-нащадок отримує властивості батька і може їх розширити
- Поліморфізм — різні класи можуть реалізовувати однаковий інтерфейс по-різному
- Абстракція — приховання деталей реалізації, робота з концепціями

```python
# ── Визначення класу ──
class Student:
    '''Клас для представлення студента.'''

    # Атрибут класу (спільний для всіх екземплярів)
    university = 'КПІ'
    _count = 0          # лічильник екземплярів

    # Конструктор: викликається при створенні об'єкта
    def __init__(self, name: str, age: int, gpa: float = 0.0):
        # Атрибути екземпляра (унікальні для кожного об'єкта)
        self.name = name          # public
        self.age  = age           # public
        self._gpa = gpa           # protected (конвенція)
        Student._count += 1

    # Метод екземпляра
    def greet(self) -> str:
        return f'Привіт! Я {self.name}, {self.age} років.'

    def __repr__(self) -> str:
        return f'Student(name={self.name!r}, age={self.age})'

# ── Створення об'єктів (екземплярів) ──
```

```python
s1 = Student('Іван', 20, 3.8)
s2 = Student('Марія', 21, 4.5)

print(s1.greet())          # Привіт! Я Іван, 20 років.
print(s1.name)             # Іван
print(Student.university)  # КПІ  — атрибут класу
print(s1.university)       # КПІ  — доступний і через екземпляр
print(Student._count)      # 2    — скільки студентів створено
print(repr(s1))            # Student(name='Іван', age=20)

# Атрибути можна додавати динамічно (але краще уникати)
s1.email = 'ivan@kpi.ua'   # Нормально, але не рекомендується
print(s1.__dict__)         # {'name': 'Іван', 'age': 20, ...}
```

Інкапсуляція: public, protected, private.
Python не має справжніх модифікаторів доступу, як Java чи C++. Замість цього використовуються конвенції іменування та механізм name mangling.

Синтаксис / Концепція	Опис / Приклад
`name`	Public. Доступний звідусіль. Звичайний атрибут або метод.
`_name`	Protected. Конвенція: «тільки для внутрішнього використання та підкласів». Python не забороняє доступ ззовні, але це сигнал розробнику.
`__name`	Private (name mangling). Python перейменовує на _ClassName__name. Захищає від випадкового перевизначення у підкласах.
`__name__`	Спеціальний. Зарезервовано Python. Не використовуйте для своїх атрибутів.

```python
class BankAccount:
    def __init__(self, owner: str, balance: float = 0.0):
        self.owner    = owner          # public: ім'я власника
        self._balance = balance        # protected: баланс
        self.__pin    = '0000'         # private: PIN-код (name mangling)
        self.__history = []            # private: історія операцій

    def deposit(self, amount: float) -> None:
        if amount <= 0:
            raise ValueError('Сума повинна бути додатною')
        self._balance += amount
        self.__history.append(f'+{amount:.2f}')

    def withdraw(self, amount: float, pin: str) -> bool:
        if pin != self.__pin:
            print('Невірний PIN!')
            return False
        if amount > self._balance:
            print('Недостатньо коштів!')
            return False
```

```
        self._balance -= amount
        self.__history.append(f'-{amount:.2f}')
        return True

    def get_history(self) -> list:
        return self.__history.copy()  # повертаємо КОПІЮ

    def __repr__(self) -> str:
        return f'BankAccount({self.owner!r}, {self._balance:.2f}₴)'

acc = BankAccount('Іван', 1000.0)
acc.deposit(500)
acc.withdraw(200, '0000')

print(acc)                      # BankAccount('Іван', 1300.00₴)
print(acc.owner)                # Іван          — public OK
print(acc._balance)             # 1300.0        — protected (можна, але
небажано)
# print(acc.__pin)              # AttributeError!
print(acc._BankAccount__pin)    # '0000' — name mangling (технічно можливо)
print(acc.get_history())        # ['+500.00', '-200.00']
```

Правило!
Використовуйте `_name` для атрибутів, що є «внутрішніми деталями реалізації».
Використовуйте `__name` лише коли треба захиститися від перевизначення у підкласах.
Для контрольованого доступу до `_атрибутів` використовуйте `@property` (розглянуто далі).

Спеціальні методи — методи Python з подвійним підкресленням на початку і кінці. Вони дозволяють класам підтримувати вбудовані операції: +, -, len(), str(), порівняння тощо.

Синтаксис / Концепція	Опис / Приклад
`__init__(self, ...)`	Конструктор. Ініціалізує новостворений об'єкт.
`__repr__(self)`	Рядкове представлення для розробника: repr(obj). Має бути однозначним.
`__str__(self)`	Рядкове представлення для користувача: str(obj), print(obj).
`__len__(self)`	Підтримка len(obj).
`__eq__(self, other)`	Оператор ==. Без нього порівнюються адреси об'єктів.
`__lt__, __le__, etc.`	Оператори <, <=, >, >= (less than, less or equal).
`__add__(self, other)`	Оператор +: obj1 + obj2.
`__getitem__(self, key)`	Підтримка obj[key].
`__contains__(self, x)`	Підтримка x in obj.
`__iter__(self)`	Підтримка ітерації: for x in obj.
`__enter__/__exit__`	Підтримка менеджера контексту with obj: ...
`__del__(self)`	Деструктор. Викликається перед знищенням об'єкта.

```python
from functools import total_ordering

@total_ordering   # автоматично генерує >, >=, <= з __eq__ і __lt__
class Vector2D:
    '''Двовимірний вектор з повним набором операцій.'''

    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def __repr__(self) -> str:              # для розробника
        return f'Vector2D({self.x}, {self.y})'

    def __str__(self) -> str:               # для користувача
        return f'({self.x}, {self.y})'

    def __add__(self, other: 'Vector2D') -> 'Vector2D':
        return Vector2D(self.x + other.x, self.y + other.y)

    def __sub__(self, other: 'Vector2D') -> 'Vector2D':
        return Vector2D(self.x - other.x, self.y - other.y)

    def __mul__(self, scalar: float) -> 'Vector2D':  # v * k
        return Vector2D(self.x * scalar, self.y * scalar)

    def __rmul__(self, scalar: float) -> 'Vector2D': # k * v
        return self.__mul__(scalar)

    def __abs__(self) -> float:             # abs(v) → довжина
        return (self.x**2 + self.y**2) ** 0.5

    def __eq__(self, other) -> bool:
        return (self.x, self.y) == (other.x, other.y)

    def __lt__(self, other) -> bool:        # порівняння за довжиною
        return abs(self) < abs(other)

    def __bool__(self) -> bool:             # нульовий вектор → False
        return self.x != 0 or self.y != 0

    def dot(self, other: 'Vector2D') -> float:  # скалярний добуток
        return self.x * other.x + self.y * other.y

v1 = Vector2D(3, 4)
v2 = Vector2D(1, 2)

print(v1)               # (3, 4)
print(repr(v1))         # Vector2D(3, 4)
print(v1 + v2)          # (4, 6)
print(v1 - v2)          # (2, 2)
print(v1 * 2)           # (6, 8)
print(3 * v2)           # (3, 6)
```

```
print(abs(v1))          # 5.0
print(v1 > v2)          # True  (5.0 > 2.23)
print(v1 == Vector2D(3, 4))  # True
print(v1.dot(v2))       # 11.0
```

`@property` перетворює метод у «розумний атрибут» — при читанні атрибута викликається getter-метод, при записі — setter, при видаленні — deleter. Це ключовий інструмент інкапсуляції в Python.

Перевага перед прямим доступом до атрибута: можна додати валідацію або обчислення без зміни інтерфейсу класу.

```
class Temperature:
    '''Клас для роботи з температурою з автоматичним перетворенням.'''

    def __init__(self, celsius: float = 0.0):
        self._celsius = celsius    # зберігаємо у захищеному атрибуті

    # — getter: читання атрибута —
    @property
    def celsius(self) -> float:
        return self._celsius

    # — setter: запис зі валідацією —
    @celsius.setter
    def celsius(self, value: float) -> None:
        if value < -273.15:
            raise ValueError(f'Нижче абсолютного нуля: {value}°C')
        self._celsius = float(value)
    # — Обчислюваний атрибут (тільки getter, без setter) —
    @property
    def fahrenheit(self) -> float:
        return self._celsius * 9/5 + 32

    @property
    def kelvin(self) -> float:
        return self._celsius + 273.15

    # — setter для fahrenheit —
    @fahrenheit.setter
    def fahrenheit(self, value: float) -> None:
        self.celsius = (value - 32) * 5/9   # через setter celsius (з
валідацією!)

    def __repr__(self) -> str:
        return (f'Temperature({self._celsius:.2f}°C | '
                f'{self.fahrenheit:.2f}°F | {self.kelvin:.2f}K)')

t = Temperature(100)
print(t)                    # Temperature(100.00°C | 212.00°F | 373.15K)

t.celsius = 37              # викликає setter
print(t.fahrenheit)         # 98.6  — читається як атрибут
```

```python
print(t.kelvin)           # 310.15

t.fahrenheit = 32         # setter fahrenheit → setter celsius
print(t.celsius)          # 0.0

# try:
#     t.celsius = -300    # ValueError!
# except ValueError as e:
#     print(e)

# ── Практичний приклад: валідований атрибут ──
class Person:
    def __init__(self, name: str, age: int):
        self.name = name
        self.age  = age      # викликає setter!

    @property
    def age(self) -> int:
        return self._age

    @age.setter
    def age(self, value: int) -> None:
        if not isinstance(value, int) or value < 0 or value > 150:
            raise ValueError(f'Некоректний вік: {value}')
        self._age = value

    @property
    def is_adult(self) -> bool:    # обчислюваний, read-only
        return self._age >= 18

p = Person('Іван', 20)
print(p.is_adult)    # True
p.age = 25           # OK
# p.age = -5         # ValueError
# p.is_adult = True  # AttributeError: can't set attribute
```

Методи екземпляра, класові та статичні

Синтаксис / Концепція	Опис / Приклад
`def method(self)`	Метод екземпляра. self — посилання на об'єкт. Має доступ до self і cls через type(self).
`@classmethod def m(cls)`	Класовий метод. cls — посилання на сам клас. Не має доступу до конкретного self. Часто використовується як альтернативний конструктор.
`@staticmethod def m()`	Статичний метод. Не отримує ані self, ані cls. Це звичайна функція у просторі імен класу — утиліта, логічно пов'язана з класом.

```python
from datetime import date

class Student:
    _registry: list = []   # реєстр усіх студентів

    def __init__(self, name: str, birth_year: int, gpa: float):
        self.name       = name
        self.birth_year = birth_year
        self._gpa       = gpa
        Student._registry.append(self)

    # ── Метод екземпляра: працює з self ──
    def info(self) -> str:
        return f'{self.name}, {self.age} р., GPA: {self._gpa}'

    @property
    def age(self) -> int:
        return date.today().year - self.birth_year

    # ── Класовий метод: альтернативний конструктор ──
    @classmethod
    def from_string(cls, data: str) -> 'Student':
        '''Створює студента з рядка "Ім\'я,рік,GPA"'''
        name, year, gpa = data.split(',')
        return cls(name.strip(), int(year), float(gpa))

    @classmethod
    def get_registry(cls) -> list:
        return cls._registry.copy()

    @classmethod
    def get_count(cls) -> int:
        return len(cls._registry)

    # ── Статичний метод: утиліта ──
    @staticmethod
    def validate_gpa(gpa: float) -> bool:
        '''Перевіряє чи GPA у діапазоні 0..5'''
        return 0.0 <= gpa <= 5.0

    @staticmethod
    def gpa_to_grade(gpa: float) -> str:
        if gpa >= 4.5: return 'Відмінно'
        if gpa >= 3.5: return 'Добре'
        if gpa >= 2.5: return 'Задовільно'
        return 'Незадовільно'

    def __repr__(self) -> str:
        return f'Student({self.name!r})'

# Звичайне створення
s1 = Student('Іван', 2004, 4.2)
```

```
# Альтернативний конструктор через classmethod
s2 = Student.from_string('Марія, 2003, 4.8')

print(s1.info())               # Іван, 21 р., GPA: 4.2
print(s2.info())               # Марія, 22 р., GPA: 4.8
print(Student.get_count())     # 2

# Статичні методи — без self і cls
print(Student.validate_gpa(3.7))   # True
print(Student.validate_gpa(6.0))   # False
print(Student.gpa_to_grade(4.8))   # Відмінно
# Можна викликати і через екземпляр (але краще через клас)
print(s1.gpa_to_grade(2.3))        # Задовільно
```

Успадкування дозволяє новому класу (нащадку) отримати атрибути та методи батьківського класу і розширити або змінити їх. В Python кожен клас неявно успадковує від вбудованого `object`.

```
class Animal:
    '''Базовий клас для всіх тварин.'''

    def __init__(self, name: str, sound: str):
        self.name  = name
        self.sound = sound
        self._is_alive = True

    def speak(self) -> str:
        return f'{self.name} каже: {self.sound}!'

    def describe(self) -> str:
        return f'Тварина: {self.name}'

    def __repr__(self) -> str:
        return f'{self.__class__.__name__}({self.name!r})'
class Dog(Animal):    # Dog успадковує Animal
    def __init__(self, name: str, breed: str):
        super().__init__(name, 'Гав')   # виклик конструктора батька
        self.breed = breed

    def fetch(self) -> str:
        return f'{self.name} приносить м\'яч!'

    # Перевизначення (override) методу батька
    def describe(self) -> str:
        parent_desc = super().describe()  # виклик батьківського методу
        return f'{parent_desc}, порода: {self.breed}'

class Cat(Animal):
    def __init__(self, name: str, indoor: bool = True):
        super().__init__(name, 'Няв')
```

```
        self.indoor = indoor

    def purr(self) -> str:
        return f'{self.name} муркоче...'

    def speak(self) -> str:       # перевизначення
        base = super().speak()
        return f'{base} (і муркоче)'

class GuideDog(Dog):   # Дворівневе успадкування
    def __init__(self, name: str, breed: str, owner: str):
        super().__init__(name, breed)
        self.owner = owner


    def guide(self) -> str:
        return f'{self.name} веде {self.owner}'

    def describe(self) -> str:
        return f'{super().describe()}, поводир для: {self.owner}'

# — Поліморфізм —
animals = [Dog('Рекс', 'Вівчарка'), Cat('Мурка'), GuideDog('Арго', 'Лабрадор',
'Іван')]

for animal in animals:
    print(animal.speak())          # у кожного свій speak()
    print(animal.describe())
    print()

# Перевірка типів
dog = Dog('Бобік', 'Такса')
print(isinstance(dog, Dog))        # True
print(isinstance(dog, Animal))     # True  — успадкований тип
print(isinstance(dog, Cat))        # False
print(issubclass(Dog, Animal))     # True
print(issubclass(GuideDog, Dog))   # True
print(issubclass(GuideDog, Animal))# True
```

Функція `super()` та порядок вирішення методів (`MRO`)/
MRO (Method Resolution Order) — порядок, у якому Python шукає метод у ієрархії класів при множинному успадкуванні. Python використовує алгоритм C3 linearization.

Переглянути MRO: `ClassName.__mro__` або `ClassName.mro()`

```
# — Множинне успадкування —
class Flyable:
    def move(self) -> str:
        return 'Летить'
    def describe(self) -> str:
        return 'може літати'
```

```python
class Swimmable:
    def move(self) -> str:
        return 'Пливе'
    def describe(self) -> str:
        return 'може плавати'

class Duck(Flyable, Swimmable):   # Flyable іде першим → його методи мають пріоритет
    def __init__(self, name: str):
        self.name = name

    def move(self) -> str:
        # super() у MRO шукає наступний клас у ланцюжку
        return f'{self.name}: {super().move()} і плаває'

duck = Duck('Кряк')
print(duck.move())          # Кряк: Летить і плаває
print(duck.describe())      # може літати  (Flyable — перший у MRO)

print(Duck.__mro__)
# (<class 'Duck'>, <class 'Flyable'>, <class 'Swimmable'>, <class 'object'>)

# ── Diamond problem та super() ──
class A:
    def hello(self):
        print('A.hello')

class B(A):
    def hello(self):
        super().hello()     # викличе C.hello (за MRO!)
        print('B.hello')

class C(A):
    def hello(self):
        super().hello()     # викличе A.hello
        print('C.hello')

class D(B, C):              # Diamond: D→B→C→A
    def hello(self):
        super().hello()
        print('D.hello')

print(D.__mro__)
# D → B → C → A → object

D().hello()
# A.hello  (спочатку A)
# C.hello  (потім C)
# B.hello  (потім B)
# D.hello  (нарешті D)
```

Правило `super()`!
Завжди використовуйте `super().__init__(...)` замість `ParentClass.__init__(self, ...)`.
Це гарантує правильну роботу MRO при множинному успадкуванні.
`super()` у методах завжди посилається на НАСТУПНИЙ клас у MRO, не обов'язково на безпосереднього батька.

Абстрактний клас — це клас, що визначає інтерфейс (набір методів), але не реалізує деякі з них. Він не може бути інстанційований напряму. Використовується для задання «контракту» для підкласів.

```python
from abc import ABC, abstractmethod
from math import pi

class Shape(ABC):     # успадковуємо ABC — Abstract Base Class
    '''Абстрактний базовий клас для геометричних фігур.'''

    def __init__(self, color: str = 'black'):
        self.color = color

    @abstractmethod
    def area(self) -> float:
        '''Обчислює площу фігури.'''
        ...

    @abstractmethod
    def perimeter(self) -> float:
        '''Обчислює периметр фігури.'''
        ...

    # Конкретний метод у абстрактному класі — дозволено
    def describe(self) -> str:
        return (f'{self.__class__.__name__} ({self.color}): '
                f'площа={self.area():.2f}, периметр={self.perimeter():.2f}')

    # Абстрактний classmethod
    @classmethod
    @abstractmethod
    def from_string(cls, s: str) -> 'Shape':
        ...

class Circle(Shape):
    def __init__(self, radius: float, color: str = 'red'):
        super().__init__(color)
        if radius <= 0:
            raise ValueError('Радіус повинен бути > 0')
        self.radius = radius

    def area(self) -> float:              # реалізуємо abstractmethod
        return pi * self.radius ** 2

    def perimeter(self) -> float:
```

```
        return 2 * pi * self.radius

    @classmethod
    def from_string(cls, s: str) -> 'Circle':
        r, color = s.split(',')
        return cls(float(r), color.strip())

    def __repr__(self) -> str:
        return f'Circle(r={self.radius}, color={self.color!r})'

class Rectangle(Shape):
    def __init__(self, width: float, height: float, color: str = 'blue'):
        super().__init__(color)
        self.width  = width
        self.height = height

    def area(self) -> float:
        return self.width * self.height

    def perimeter(self) -> float:
        return 2 * (self.width + self.height)

    @classmethod
    def from_string(cls, s: str) -> 'Rectangle':
        w, h, color = s.split(',')
        return cls(float(w), float(h), color.strip())

    def __repr__(self) -> str:
        return f'Rectangle({self.width}x{self.height})'

# shape = Shape()   # TypeError: Can't instantiate abstract class!

shapes: list[Shape] = [
    Circle(5, 'green'),
    Rectangle(4, 6, 'blue'),
    Circle.from_string('3, red'),
    Rectangle.from_string('10, 2, black'),
]

# Поліморфізм через спільний інтерфейс
for shape in shapes:
    print(shape.describe())

# Сортування за площею
sorted_shapes = sorted(shapes, key=lambda s: s.area())
print('\nВід меншої до більшої площі:')
for s in sorted_shapes:
    print(f'  {s}: {s.area():.2f}')
```

Декоратор `@dataclass` автоматично генерує `__init__`, `__repr__`, `__eq__` та інші методи на основі оголошених атрибутів. Ідеальний для класів-контейнерів даних.

```python
from dataclasses import dataclass, field
from typing import ClassVar

@dataclass
class Point:
    x: float
    y: float

    def distance_to(self, other: 'Point') -> float:
        return ((self.x-other.x)**2 + (self.y-other.y)**2)**0.5

p1 = Point(1.0, 2.0)
p2 = Point(4.0, 6.0)
print(p1)                   # Point(x=1.0, y=2.0)  — автогенерований __repr__
print(p1 == Point(1.0, 2.0))  # True  — автогенерований __eq__
print(p1.distance_to(p2))  # 5.0

@dataclass(order=True)      # генерує <, <=, >, >= на основі порядку полів
class Student:
    # sort_index для сортування (не відображається у repr за замовч.)
    gpa:  float = field(compare=True)
    name: str   = field(compare=False)
    grades: list = field(default_factory=list, compare=False)

    # Атрибут класу (не екземпляра)
    _count: ClassVar[int] = 0

    def __post_init__(self):
        '''Викликається після автогенерованого __init__'''
        if not (0 <= self.gpa <= 5):
            raise ValueError(f'Невалідний GPA: {self.gpa}')
        Student._count += 1

students = [
    Student(4.5, 'Марія'),
    Student(3.8, 'Іван'),
    Student(4.9, 'Олена'),
]

# Сортування за GPA (order=True)
students.sort(reverse=True)
for s in students:
    print(s)
# Student(gpa=4.9, name='Олена', grades=[])
# Student(gpa=4.5, name='Марія', grades=[])
# Student(gpa=3.8, name='Іван',  grades=[])
```

Найкращі практики. `__slots__`, `__all__`, протоколи

`__slots__`: оптимізація пам'яті

```python
# За замовчуванням атрибути зберігаються у __dict__ (словник)
# __slots__ замінює словник на фіксований набір — менше пам'яті

class PointSlots:
    __slots__ = ('x', 'y')   # дозволяємо лише ці атрибути

    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

p = PointSlots(1.0, 2.0)
# p.z = 3.0   # AttributeError: 'PointSlots' has no attribute 'z'
# print(p.__dict__)  # AttributeError: немає __dict__!

# Для великої кількості об'єктів __slots__ економить ~40% пам'яті
```

Протокол `Iterator`: власний ітерований клас

```python
class CountDown:
    '''Ітерований зворотній відлік.'''

    def __init__(self, start: int):
        self._start = start

    def __iter__(self):
        self._current = self._start
        return self

    def __next__(self):
        if self._current < 0:
            raise StopIteration
        value = self._current
        self._current -= 1
        return value

    def __len__(self):
        return self._start + 1

    def __contains__(self, item):
        return 0 <= item <= self._start

cd = CountDown(5)
print(list(cd))          # [5, 4, 3, 2, 1, 0]
print(len(cd))           # 6
print(3 in cd)           # True
for n in CountDown(3):   # 3 2 1 0
    print(n, end=' ')
```

Менеджер контексту через клас

```python
class ManagedFile:
    '''Власний менеджер контексту для файлу.'''

    def __init__(self, filename: str, mode: str = 'r'):
        self.filename = filename
        self.mode     = mode
        self._file    = None

    def __enter__(self):
        print(f'Відкриваємо: {self.filename}')
        self._file = open(self.filename, self.mode, encoding='utf-8')
        return self._file

    def __exit__(self, exc_type, exc_val, exc_tb):
        if self._file:
            self._file.close()
        print(f'Закрито: {self.filename}')
        # Повертаємо False: не пригнічуємо виняток
        return False

with ManagedFile('data.txt', 'w') as f:
    f.write('Тест\n')
# Відкриваємо: data.txt
# Закрито: data.txt
```

Комплексний приклад: система управління бібліотекою.

Розглянемо повноцінний приклад застосування всіх концепцій ООП у реальному сценарії.

```python
from abc import ABC, abstractmethod
from dataclasses import dataclass, field
from datetime import date, timedelta
from typing import Optional

# ── Абстрактний базовий клас ──
class LibraryItem(ABC):
    _total_items: int = 0

    def __init__(self, title: str, year: int):
        self.title      = title
        self.year       = year
        self._available = True
        LibraryItem._total_items += 1

    @abstractmethod
    def get_info(self) -> str: ...

    @abstractmethod
    def loan_period_days(self) -> int: ...
```

```
    @property
    def available(self) -> bool:
        return self._available

    def checkout(self) -> bool:
        if not self._available:
            return False
        self._available = False
        return True

    def return_item(self) -> None:
        self._available = True

    @classmethod
    def total_items(cls) -> int:
        return cls._total_items

    def __repr__(self) -> str:
        status = 'доступна' if self._available else 'видана'
        return f'{self.__class__.__name__}({self.title!r}) [{status}]'

class Book(LibraryItem):
    def __init__(self, title: str, author: str, year: int, pages: int):
        super().__init__(title, year)
        self.author = author
        self.pages  = pages

    def get_info(self) -> str:
        return f'Книга: "{self.title}" — {self.author} ({self.year},
{self.pages} стор.)'

    def loan_period_days(self) -> int:
        return 14   # 2 тижні

class DVD(LibraryItem):
    def __init__(self, title: str, director: str, year: int, duration_min:
int):
        super().__init__(title, year)
        self.director     = director
        self.duration_min = duration_min

    def get_info(self) -> str:
        return f'DVD: "{self.title}" — реж. {self.director}
({self.duration_min} хв.)'

    def loan_period_days(self) -> int:
        return 3    # 3 дні

@dataclass
class Loan:
    item:      LibraryItem
    borrower:  str
```

```
    loan_date: date = field(default_factory=date.today)

    @property
    def due_date(self) -> date:
        return self.loan_date + timedelta(days=self.item.loan_period_days())

    @property
    def is_overdue(self) -> bool:
        return date.today() > self.due_date

    def __str__(self) -> str:
        status = ' [ПРОСТРОЧЕНО!]' if self.is_overdue else ''
        return (f'{self.borrower}: "{self.item.title}"'
                f' до {self.due_date}{status}')

class Library:
    def __init__(self, name: str):
        self.name    = name
        self._items: list[LibraryItem] = []
        self._loans: list[Loan]        = []

    def add_item(self, item: LibraryItem) -> None:
        self._items.append(item)

    def checkout(self, title: str, borrower: str) -> Optional[Loan]:
        for item in self._items:
            if item.title == title and item.available:
                item.checkout()
                loan = Loan(item, borrower)
                self._loans.append(loan)
                return loan
        return None

    def return_item(self, title: str) -> bool:
        for loan in self._loans:
            if loan.item.title == title and not loan.item.available:
                loan.item.return_item()
                self._loans.remove(loan)
                return True
        return False

    def available_items(self) -> list[LibraryItem]:
        return [i for i in self._items if i.available]

    def report(self) -> None:
        print(f'=== Бібліотека "{self.name}" ===')
        print(f'Всього одиниць: {len(self._items)}')
        print(f'Доступних:      {len(self.available_items())}')
        print(f'Виданих:        {len(self._loans)}')
        if self._loans:
            print('Активні позики:')
            for loan in self._loans:
```

```
                print(f'  {loan}')

# ── Використання ──
lib = Library('Міська бібліотека')

lib.add_item(Book('Кобзар', 'Т. Шевченко', 1840, 312))
lib.add_item(Book('Тіні забутих предків', 'М. Коцюбинський', 1911, 128))
lib.add_item(DVD('Поводир', 'О. Саніна', 2013, 112))

loan1 = lib.checkout('Кобзар', 'Іван Петренко')
loan2 = lib.checkout('Поводир', 'Марія Коваль')

lib.report()

lib.return_item('Кобзар')
print('\nПісля повернення:')
lib.report()
```

Завдання на лабораторну роботу

Кожне завдання оформляйте у окремому файлі (task1.py ... task7.py). Для класів обов'язково пишіть docstrings. Використовуйте анотації типів.

8.1. Клас з атрибутами, методами та `@property`. Розробіть клас `Rectangle` (прямокутник):

- конструктор приймає `width` та `height`. Обидва — protected через `@property` з валідацією (> 0);
- Properties (read-only): `area`, `perimeter`, `diagonal`, `is_square`;
- методи: `scale(factor)` — повертає новий `Rectangle` із масштабованими розмірами; `describe()` — форматований рядок опису;
- спеціальні методи: `__repr__`, `__str__`, `__eq__` (рівність за розмірами), `__lt__` (порівняння за площею), `__add__` (повертає прямокутник із сумарною площею, що має ту ж ширину);
- Класовий метод `from_string('4x5')` та статичний метод `is_valid(w, h)`.

Напишіть мінімум 10 тестових виразів, що демонструють усі можливості класу.

8.2. Інкапсуляція. Public, protected, private. Розробіть клас `Student` із такою структурою атрибутів:

- Public: `name`, `year` (рік навчання 1–6);
- Protected: `_gpa`, `_courses` (список курсів);
- Private: `__student_id` (автогенерований номер: `'STU-001'`, `'STU-002'` тощо);
- `@property` з getter та setter для `gpa` (валідація 0–5) та `year` (валідація 1–6);
- методи: `add_course(name)`, `remove_course(name)`, `get_transcript()` → форматована відомість;
- класові методи: `get_count()`, `find_by_id(id)` → повертає студента з реєстру;
- статичний метод: `validate_name(name)` → перевіряє що ім'я складається лише з літер.

Продемонструйте спробу прямого доступу до `__student_id` та порівняйте з правильним способом.

8.3. Успадкування та поліморфізм.

Розробіть ієрархію класів для системи транспортних засобів:

- Базовий клас `Vehicle`: `make`, `model`, `year`, `_mileage`; методи `drive(km)`, `refuel(liters)`, `status()`; `@property mileage` (read-only); `__repr__`;
- Клас `Car(Vehicle)`: додає `fuel_type`, `fuel_level`; перевизначає `drive()` (зменшує паливо); додає `property range` (запас ходу);
- Клас `ElectricCar(Car)`: замість палива — `battery_level` (%); перевизначає `drive()` та `refuel()` (перейменувати на `charge()`);
- Клас `Truck(Vehicle)`: вантажопідйомність, поточна вага вантажу; методи `load(kg)` та `unload(kg)` з перевіркою;
- Клас `FleetManager`: зберігає список `Vehicle`; методи `add_vehicle`, `remove_vehicle`, `total_mileage`, `find_by_make(make)`, `report()`.

Створіть флот з 5–6 різних автомобілів. Продемонструйте поліморфізм: для кожного транспорту викличте `drive(100)` — у кожного своя логіка.

Контрольні запитання

1. Що таке клас і об'єкт у Python? Поясніть різницю між атрибутом класу та атрибутом екземпляра на конкретному прикладі.

2. Поясніть механізм інкапсуляції в Python. Яка різниця між `_name` та `__name`? Що таке name mangling і навіщо він потрібен?

3. Що таке `@property`? Як реалізувати атрибут лише для читання (read-only) через `@property`? Яку проблему вирішує `@property` порівняно з прямим доступом до атрибута?

4. У чому різниця між методом екземпляра, `@classmethod` та `@staticmethod`? Коли доцільно використовувати кожен з них?

5. Що таке `__init__` і `__new__`? Що таке спеціальні методи? Назвіть п'ять спеціальних методів, поясніть які вбудовані операції вони підтримують.

6. Поясніть механізм успадкування. Для чого використовується `super()`? Що відбудеться, якщо замінити `super().__init__()` на `ParentClass.__init__(self)`?

7. Що таке MRO (Method Resolution Order)? За яким алгоритмом Python визначає порядок пошуку методів при множинному успадкуванні? Яку послідовність класів поверне `Dog.__mro__` для class `Dog(Flyable, Swimmable)`?

8. Що таке абстрактний клас? Навіщо використовується модуль `abc`? Що станеться, якщо спробувати створити екземпляр класу, що успадковує ABC, але не реалізує всі `@abstractmethod`?

9. Що таке `@dataclass`? Які спеціальні методи він генерує автоматично? Навіщо потрібен `__post_init__`?

10. Що таке `__slots__`? Яку перевагу дає його використання? В яких ситуаціях воно особливо корисне?