**ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ  
«ОПТИКО-МЕХАНІЧНИЙ ФАХОВИЙ КОЛЕДЖ КИЇВСЬКОГО  
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ІМЕНІ ТАРАСА ШЕВЧЕНКА»**Циклова комісія програмування та інформаційних технологій

**ЗВІТ**

**З НАВЧАЛЬНОЇ ПРАКТИКИ**

спеціальність *121 Інженерія програмного забезпечення*освітньо-кваліфікаційний рівень *«Фаховий молодший бакалавр»*

| ОЦІНКА  (оцінка цифрою та прописом)  Керівник практики від коледжу:  (підпис)  « \_» 20 року | Виконав:  студентка 3-го курсу групи ІПЗ-32  *Ярошенко Вікторія Олександрівна*  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис виконавця)  « \_» 20 року |
| --- | --- |

КИЇВ — 2025

ЗАТВЕРДЖУЮ

Директор ВСП «ОМФК КНУ

ім. Тараса Шевченка»

\_\_\_\_\_\_\_\_\_\_\_Борис ГАПРИНДАШВІЛІ

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025р

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

на період *Навчальної* практики

(навчальної, технологічної, виробничої)

*Ярошенко Вікторія Олександрівна*

(П І.Б студента )

Спеціальність *121 Інженерія програмного забезпечення* Група *ІПЗ-32*

|  | **Питання для вивчення** | **Відмітки про виконання** |
| --- | --- | --- |
| **1** | Вступне заняття. Вивчення питань охорони праці. Ознайомлення з електронною бібліотекою коледжу. Вивчення правил оформлення звітної документації. |  |
| **2** | Знайомство з мовою програмування Python. Історія мови, особливості мови, сфери застосування мови |  |
| 3 | Основи мови Python. Синтаксис мови, середовище виконання, запуск програм |  |
| 4 | Змінні та літерали, типи даних. Вбудовані типи даних, ініціалізація змінних, змінювані і незмінювані типи |
| 5 | Програмування лінійних та розгалужених обчислювальних процесів |  |
| 6 | Програмування циклічних обчислювальних процесів |  |
| 7 | Типи даних визначені користувачем, магічні методи, виключення (Exceptions) та їх обробка |  |
| 8 | Вбудовані структури даних, складні типи |  |
| 9 | Функції |  |
| 10 | Потокове введення, виведення та обробка даних |  |
| 11 | Побудова проекту, модулі, пакети |  |
| 12 | Робота з файловою системою, оператор with, контекстний менеджер |  |
| 13 | Робота з протоколом HTTP, бібліотека requests |  |

| Голова циклової комісії |  |  |  |
| --- | --- | --- | --- |
|  | (підпис) | (ініціали, прізвище) | (дата) |
| Керівник практики від коледжу |  |  |  |
|  | (підпис) | (ініціали, прізвище) | (дата) |

**Зміст**

[**ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 2**](#_heading=h.gjdgxs)

[**Вступ 4**](#_heading=h.30j0zll)

[1. Теоретичні відомості 5](#_heading=h.moj9lv3fbjcp)

[2. Програмування лінійних та розгалужених обчислювальних процесів 9](#_heading=h.d3x43kxoz8ba)

[3. Функції 13](#_heading=h.lcb9halyqp59)

[4. Робота з файловою системою, оператор with та контекстні менеджери 15](#_heading=h.2bhsxsp9hqdg)

[**5 Виконання завдань 19**](#_heading=h.1ksv4uv)

[**5.1 Завдання 1 19**](#_heading=h.44sinio)

[**5.2 Завдання 2 22**](#_heading=h.2jxsxqh)

[**5.3 Завдання 3 40**](#_heading=h.z337ya)

[**6 Висновки 44**](#_heading=h.3j2qqm3)

[**ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ІНФОРМАЦІЇ 45**](#_heading=)

# **ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ**

Перелік скорочень, умовних позначень, символів, одиниць і термінів (Python):

1. API — Application Programming Interface (Інтерфейс програмування додатків).
2. IDE — Integrated Development Environment (Інтегроване середовище розробки).
3. CLI — Command Line Interface (Інтерфейс командного рядка).
4. OS — Operating System (Операційна система).
5. GUI — Graphical User Interface (Графічний інтерфейс користувача).
6. JSON — JavaScript Object Notation (формат обміну даними, що використовується для зберігання та передачі структурованої інформації).
7. HTML — HyperText Markup Language (Мова розмітки гіпертексту).
8. CSS — Cascading Style Sheets (Каскадні таблиці стилів).
9. XML — Extensible Markup Language (Мова розмітки розширюваних документів).
10. SQL — Structured Query Language (Мова запитів до бази даних).
11. GUI — Graphical User Interface (Графічний інтерфейс користувача).
12. CSV — Comma-Separated Values (формат файлів з розділенням даних комами).
13. DB — Database (База даних).
14. RAM — Random Access Memory (Оперативна пам'ять).
15. CPU — Central Processing Unit (Центральний процесор).
16. URL — Uniform Resource Locator (Уніфікований локатор ресурсу).
17. HTTP — Hypertext Transfer Protocol (Протокол передачі гіпертексту).
18. HTTPS — Hypertext Transfer Protocol Secure (Захищений протокол передачі гіпертексту).
19. IDE — Integrated Development Environment (Інтегроване середовище розробки).
20. IDE — Integrated Development Environment (Інтегроване середовище розробки).

Умовні позначення та символи:

* = — Оператор присвоєння.
* == — Оператор порівняння на рівність.
* != — Оператор порівняння на нерівність.
* > — Оператор порівняння (більше).
* < — Оператор порівняння (менше).
* >= — Оператор порівняння (більше або рівне).
* <= — Оператор порівняння (менше або рівне).
* and — Логічний оператор "і".
* or — Логічний оператор "або".
* not — Логічний оператор "не".
* is — Оператор порівняння для перевірки ідентичності.
* in — Оператор перевірки наявності елемента у колекції.

Типи даних в Python:

* int — Ціле число.
* float — Число з плаваючою комою.
* str — Рядок (текст).
* bool — Логічне значення (True або False).
* list — Список.
* tuple — Кортеж.
* set — Множина.
* dict — Словник (словникове співвідношення ключ-значення).

Одиниці вимірювання часу (під час роботи з датами та часом):

* s — секунди.
* ms — мілісекунди.
* us — мікросекунди.
* ns — наносекунди.

Терміни в Python:

* Модуль — Файл з Python-кодом, який може містити функції, класи та змінні.
* Пакет — Колекція модулів, яка містить код для виконання певної задачі.
* Ітератор — Об'єкт, який дозволяє по черзі отримувати елементи з колекції.
* Генератор — Спеціальний тип ітератора, що генерує значення за потребою, замість збереження всіх значень у пам'яті.
* Декоратор — Функція, яка модифікує інші функції або методи.
* Lambda-функція — Анонімна функція, яка визначається без використання ключового слова def.

# **Вступ**

Автор: Ярошенко Вікторія Олександрівна

Дата: 23.01.2025

Мета: Узагальнення вивчених тем

Перелік вивчених тем:

1. Вступне заняття. Вивчення питань охорони праці. Ознайомлення з електронною бібліотекою коледжу. Вивчення правил оформлення звітної документації.
2. Знайомство з мовою програмування Python. Історія мови, особливості мови, сфери застосування мови.
3. Основи мови Python. Синтаксис мови, середовище виконання, запуск програм.
4. Змінні та літерали, типи даних. Вбудовані типи даних, ініціалізація змінних, змінювані і незмінювані типи.
5. Програмування лінійних та розгалужених обчислювальних процесів.
6. Програмування циклічних обчислювальних процесів
7. Типи даних визначені користувачем, магічні методи, виключення (Exceptions) та їх обробка.
8. Вбудовані структури даних, складні типи
9. Функції.
10. Потокове введення, виведення та обробка даних.
11. Побудова проекту, модулі, пакети.
12. Робота з файловою системою, оператор with, контекстний менеджер.
13. Робота з протоколом HTTP, бібліотека requests.
14. Створення HTTP API, бібліотека FastAPI.

### **1. Теоретичні відомості**

#### **1.1 Вступне заняття**

На першому занятті було детально розглянуто питання безпеки праці, проведено інструктаж, який гарантує безпечні умови як для навчання, так і для роботи. Також було ознайомлено з можливостями та правилами користування електронною бібліотекою коледжу, що сприяє ефективному пошуку та використанню навчальних матеріалів. Крім того, розглянуто вимоги до оформлення звітної документації, що є важливим для правильного ведення навчальних робіт.

#### **1.2 Знайомство з мовою програмування Python**

Python – це високорівнева інтерпретована мова програмування, створена Гвідо ван Россумом у 1991 році. Основна ідея при розробці полягала у спрощенні читання коду та підвищенні продуктивності розробки. Назва мови пов’язана з популярним британським комедійним гуртом "Monty Python’s Flying Circus". Завдяки простому синтаксису, гнучкості та активній підтримці спільноти, Python здобув широку популярність.

Основні характеристики Python:

* **Динамічна типізація:** тип змінної визначається під час виконання програми.
* **Автоматичне управління пам’яттю:** використовується збирач сміття для автоматичного звільнення невикористовуваної пам’яті.
* **Лаконічний синтаксис:** код виглядає просто і читається як звичайний текст.
* **Кросплатформеність:** мова працює на Windows, Linux, macOS та інших операційних системах.

Python дозволяє виконувати операції між різними типами даних без явного приведення типів, що обумовлено його слабкою типізацією, при цьому підтримується динамічна типізація.

Сфери застосування Python:

* **Web-розробка:** використання фреймворків, таких як Django та Flask.
* **Інтернет речей (IoT):** зручність програмування мікроконтролерів, наприклад, Raspberry Pi.
* **Data Science та машинне навчання:** аналіз даних та побудова моделей завдяки бібліотекам NumPy, Pandas, TensorFlow, Scikit-learn.
* **Системне програмування:** автоматизація адміністрування систем і написання скриптів.
* **DevOps:** створення автоматизованих скриптів для керування інфраструктурою, розгортання серверів і роботи з хмарними сервісами.

#### **1.3 Основи мови Python**

Python характеризується зрозумілим та компактним синтаксисом. Наприклад, для виведення повідомлення "Hello, World!" достатньо написати:

python

КопироватьРедактировать

print("Hello, World!")

Основні елементи мови включають:

* **Алфавіт:** набір символів (латинські літери, цифри, спеціальні символи, пробіли, табуляції).
* **Лексеми:** найменші одиниці коду, такі як ключові слова, ідентифікатори, оператори та роздільники.
* **Вирази:** комбінації значень, змінних та операторів, що обчислюються у певний результат.

##### **Середовище виконання**

Python-програми можуть виконуватись у різних середовищах: інтегрованих середовищах розробки (IDE), через термінал або командний рядок. Основним інтерпретатором є CPython, проте існують альтернативи:

* **PyPy:** використовує JIT-компіляцію для підвищення швидкодії.
* **Jython:** версія для Java-платформи, що забезпечує інтеграцію з Java.
* **IronPython:** реалізація для .NET, яка дозволяє інтегруватися з C# та іншими мовами платформи.

Python – це інтерпретована мова, що виконує код построчно, на відміну від мов, де код попередньо компілюється у виконуваний файл.

##### **Кросплатформеність**

Завдяки своїй кросплатформеності Python дозволяє запускати однаковий код на різних операційних системах за умови наявності відповідного інтерпретатора.

#### **1.4 Змінні, літерали та типи даних**

Python використовує динамічну типізацію, тому оголошувати тип змінної заздалегідь не потрібно – він визначається автоматично при присвоєнні значення:

python

КопироватьРедактировать

x = 10 # ціле число

pi = 3.14 # число з плаваючою точкою

name = "Anna" # рядок

##### **Вбудовані типи даних**

Розрізняють змінювані та незмінювані типи:

* **Незмінювані:** int, float, str, tuple, bool.
* **Змінювані:** list, dict, set.

У Python змінні є посиланнями на об'єкти в пам’яті. Наприклад:

python

КопироватьРедактировать

lst1 = [1, 2, 3]

lst2 = lst1 # lst2 посилається на той самий список

lst2.append(4)

print(lst1) # виведе [1, 2, 3, 4]

##### **Обчислення та пріоритет операцій**

Приклади обчислень:

python

КопироватьРедактировать

result = 2 + 3 \* 4 # результат: 14, оскільки множення має вищий пріоритет

result = (2 + 3) \* 4 # результат: 20, дужки змінюють порядок виконання

Python має вбудовані функції для числових обчислень, наприклад:

* abs()
* round()
* pow()
* max()

Також модуль math надає розширені математичні можливості.

### **2. Програмування лінійних та розгалужених обчислювальних процесів**

Python дозволяє створювати як лінійні, так і розгалужені алгоритми. Лінійний код виконується послідовно, а розгалужений – залежно від виконання певних умов.

#### **Логічні оператори та умовні конструкції**

Для формування умов використовуються логічні оператори:

* **and:** обидві умови мають бути істинними.
* **or:** хоча б одна умова має бути істинною.
* **not:** інвертує логічне значення.

Приклад:

python

КопироватьРедактировать

x = 10

y = 20

print(x > 5 and y < 30) # True

print(x < 5 or y > 30) # False

print(not (x > 5)) # False

Конструкції if-elif-else дозволяють реалізувати розгалуження:

python

КопироватьРедактировать

age = 18

if age < 18:

print("Молодший за 18")

elif age == 18:

print("Саме 18 років")

else:

print("Старший за 18")

Також існує тернарний оператор для скороченого запису умовних виразів:

python

КопироватьРедактировать

min\_value = x if x < y else y

Відступи в Python мають важливе значення, оскільки вони визначають блоки коду, зокрема у вкладених умовах.

#### **2.1 Програмування циклічних процесів**

Цикли дозволяють виконувати повторювані операції. Основні конструкції:

* **for:** використовується для перебору елементів ітерабельних об'єктів.
* **while:** виконується, поки умова залишається істинною.

Приклади:

python

КопироватьРедактировать

for i in range(5):

print(i)

x = 0

while x < 5:

print(x)

x += 1

##### **Ітератори та генератори**

* **Ітератори:** об'єкти, що реалізують методи \_\_iter\_\_() та \_\_next\_\_(), забезпечують послідовний доступ до елементів.
* **Генератори:** спеціальний тип ітераторів, які використовують ключове слово yield для збереження стану між викликами. Вони економлять пам’ять, оскільки генерують елементи «на льоту».

Приклад генератора:

python

КопироватьРедактировать

def my\_generator():

for i in range(5):

yield i

for value in my\_generator():

print(value)

Генератори та ітератори особливо корисні при обробці великих обсягів даних, наприклад, при читанні великих файлів.

#### **2.2 Типи даних, визначені користувачем, магічні методи та обробка виключень**

Python дозволяє створювати власні типи даних за допомогою класів. Клас є шаблоном, що описує властивості та поведінку об'єктів. Це сприяє кращій організації коду.

**Магічні методи** (методи з подвійним підкресленням, наприклад, \_\_init\_\_, \_\_str\_\_, \_\_eq\_\_) дозволяють змінювати стандартну поведінку об'єктів, роблячи класи більш інтуїтивними.

**Обробка виключень** здійснюється через конструкції try-except, що дозволяють коректно реагувати на помилки під час виконання програми. Можна також використовувати блоки else та finally для виконання додаткових дій. Крім того, Python дозволяє створювати власні класи виключень, наслідуючи базовий клас Exception.

#### **2.3 Вбудовані структури даних та складні типи**

Python містить різноманітні структури даних:

* **Списки (list):** динамічні структури, які можуть містити елементи різних типів із можливістю індексації.
* **Кортежі (tuple):** незмінні послідовності, що забезпечують швидкий доступ до елементів і можуть бути ключами у словниках.
* **Словники (dict):** реалізовані за допомогою хеш-таблиць, що дозволяють швидко знаходити значення за ключем.
* **Множини (set):** зберігають унікальні елементи та підтримують операції об’єднання, перетину та різниці.
* **Рядки (str):** незмінні послідовності символів.

##### **Хеш-функції та хеш-таблиці**

Хеш-функції перетворюють дані у числові значення (хеш), що використовуються у хеш-таблицях для швидкого доступу до даних у форматі «ключ-значення». Дані, що використовуються як ключі, мають бути незмінними та відповідати певним вимогам.

##### **Реалізація списків**

Списки в Python реалізовані як масиви об’єктів (PyObject), що дозволяє швидко додавати та видаляти елементи, хоча може призводити до підвищеного споживання пам’яті. У порівнянні, зв’язані списки, що використовуються в інших мовах, ефективні для частих змін, але повільніші у доступі до елементів.

Правильний вибір структури даних відіграє ключову роль у підвищенні ефективності алгоритмів.

### **3. Функції**

Функції допомагають структурувати код, уникати дублювання та спрощують його підтримку. Вони створюються за допомогою ключового слова def з подальшим зазначенням імені та параметрів:

python

КопироватьРедактировать

def назва\_функції(параметри):

# тіло функції

return значення

Функції можуть приймати параметри, що дозволяє передавати необхідні дані, і повертати результати за допомогою оператора return.

#### **Передача параметрів та область видимості**

Python передає параметри за посиланням, тому функція отримує доступ до оригінальних даних. При цьому локальні змінні функції не впливають на глобальні, якщо не використовувати спеціальні ключові слова (global, nonlocal).

#### **Функції як об’єкти**

Функції у Python є об’єктами першого класу – їх можна передавати як аргументи іншим функціям, зберігати у змінних або повертати як результат виконання інших функцій.

#### **Анонімні функції (lambda) та рекурсія**

Для створення коротких функцій, що складаються з одного виразу, використовуються lambda-функції:

python

КопироватьРедактировать

lambda аргументи: вираз

Рекурсія – це виклик функції самою собою, що може бути корисним для вирішення задач з деревоподібними структурами, хоча варто враховувати обмеження на глибину рекурсії.

#### **3.1 Потокове введення, виведення та обробка даних**

Робота з потоками вводу/виводу є важливою при обробці великих обсягів даних у режимі реального часу. Python використовує:

* **sys.stdin** для вводу даних.
* **sys.stdout** для стандартного виводу.
* **sys.stderr** для виведення повідомлень про помилки.

Цей механізм дозволяє обробляти дані «на льоту», що корисно при аналізі логів або роботі з потоковими даними.

Також використовуються різні формати для зберігання даних:

* **JSON Lines (NDJSON):** кожен JSON-об’єкт записується в окремому рядку.
* **Protocol Buffers (Protobuf):** ефективне бінарне представлення структурованих даних.
* **Parquet:** колонковий формат, зручний для аналітики.
* **CSV:** текстовий формат для табличних даних.

Окрім цього, за допомогою os.environ можна керувати змінними оточення для налаштування програми без зміни вихідного коду.

#### **3.2 Побудова проекту: модулі та пакети**

Щоб уникнути збирання всього коду в одному файлі, Python дозволяє розділяти його на модулі. Наприклад, функції калькулятора можна розмістити в окремому файлі math\_operations.py. Коли модулів багато, їх організують у пакети – папки зі спеціальним файлом \_\_init\_\_.py, що вказує Python, що ця папка є пакетом. Структура проекту може виглядати наступним чином:

css

КопироватьРедактировать

my\_project/

│── main.py

│── my\_package/

│── \_\_init\_\_.py

│── module1.py

│── module2.py

Python також має безліч готових бібліотек, які встановлюються через менеджер пакетів pip (наприклад, бібліотека requests для роботи з HTTP).

### **4. Робота з файловою системою, оператор with та контекстні менеджери**

При роботі з файлами важливо правильно управляти ресурсами. Наприклад, для читання текстового файлу можна використовувати стандартний метод відкриття, читання та закриття файлу:

python

КопироватьРедактировать

file = open("data.txt", "r")

data = file.read()

file.close()

Однак, у разі виникнення помилки файл може залишитись відкритим. Для уникнення такої ситуації використовують оператор with, який автоматично закриває файл після завершення роботи з ним:

python

КопироватьРедактировать

with open("data.txt", "r") as file:

data = file.read()

Цей підхід робить роботу з файлами безпечнішою. Оператор with можна застосовувати не лише для файлів, а й для створення власних контекстних менеджерів за допомогою методів \_\_enter\_\_ та \_\_exit\_\_:

python

КопироватьРедактировать

class FileManager:

def \_\_init\_\_(self, filename, mode):

self.file = open(filename, mode)

def \_\_enter\_\_(self):

return self.file

def \_\_exit\_\_(self, exc\_type, exc\_value, traceback):

self.file.close()

with FileManager("data.txt", "r") as file:

data = file.read()

#### **4.1 Робота з протоколом HTTP та бібліотекою requests**

HTTP – це протокол, що визначає правила обміну даними між клієнтом і сервером. При роботі з веб-ресурсами використовуються різні методи запитів:

* **GET:** отримання даних.
* **POST:** надсилання нових даних.
* **PUT:** оновлення даних.
* **DELETE:** видалення даних.

Бібліотека requests дозволяє легко виконувати HTTP-запити. Наприклад, для GET-запиту:

python

КопироватьРедактировать

import requests

response = requests.get("https://jsonplaceholder.typicode.com/posts/1")

print(response.status\_code) # виводить код відповіді

print(response.json()) # отримання даних у форматі JSON

Важливо перевіряти статус-код відповіді, щоб впевнитися в успішному виконанні запиту:

python

КопироватьРедактировать

if response.status\_code == 200:

print("Успішний запит")

else:

print("Помилка")

Також можна передавати параметри запиту та використовувати сесії для збереження стану між запитами:

python

КопироватьРедактировать

params = {"userId": 1}

response = requests.get("https://jsonplaceholder.typicode.com/posts", params=params)

session = requests.Session()

session.get("https://example.com/login") # виконання логіну

response = session.get("https://example.com/dashboard") # подальше використання сесії

# **5 Виконання завдань**

## **5.1 Завдання 1**

*Написати функцію, яка:*

приймає на вхід строку і повертає кортеж з трьома елементами, в яких перший елемент має тип str і включає всі голосні літери з вхідної строки (зворотній алфавітний порядок). Другий елемент - int - кількість голосних літер, а останній - всі приголосні літери з вхідної строки (зворотній алфавітний порядок).

Приклад:

вхідна строка: abcdefg

вихід: (ea, 2, gfdcb)

**Пояснення:**

Цей код здійснює наступні операції:

1. Зчитує вхідний рядок і визначає його довжину.
2. Ініціалізує три змінні:
   * line1 для накопичення голосних літер;
   * line2 для підрахунку кількості голосних літер (початкове значення 0);
   * line3 для накопичення приголосних літер.
3. Визначає два списки:
   * один містить голосні літери (letters\_1);
   * інший містить приголосні літери (letters\_2).
4. Для формування першого елемента кортежу (рядок, що містить голосні):
   * Проходить по списку голосних у зворотному порядку (тобто починаючи з останнього елемента і до першого).
   * Для кожної голосної виконує перебір усіх символів вхідного рядка.
   * Якщо символ рядка співпадає з розгляданою голосною, додає цю голосну до рядка line1 та збільшує лічильник голосних (line2) на 1.
5. Для формування третього елемента кортежу (рядок, що містить приголосні):
   * Аналогічно, перебирає список приголосних у зворотному порядку.
   * Для кожного символу з цього списку виконує перебір усіх символів вхідного рядка.
   * Якщо символ рядка співпадає з поточною приголосною, додає її до рядка line3.
6. Нарешті, виводить кортеж, де:
   * перший елемент – рядок з голосними (в зворотному алфавітному порядку),
   * другий елемент – число, що відповідає кількості голосних,
   * третій елемент – рядок з приголосними (також у зворотному алфавітному порядку).

**Код:**

string = input()

size = len(string)

string\_changed=string

line1=""

line2=0

line3=""

letters\_1=['a','e','i','o','u','y']

letters\_2=['b','c','d','f','g','h','j','k','l','m','n','p','q','r','s','t','v','w','x','z']

for i in range(5,-1,-1):

for y in range(0,size):

if string[y]==letters\_1[i]:

line1+=(letters\_1[i])

line2+=1

for i1 in range(len(letters\_2)-1,-1,-1):

for y1 in range(0,size):

if string[y1]==letters\_2[i1]:

line3+=(letters\_2[i1])

print("("+line1+ ", " +str(line2)+ ", "+line3+")")

## **5.2 Завдання 2**

Список задач для покращення навичок з програмування та вивчення Python зокрема.

**Пояснення:**

1. Піфагорові трійки

Завдання: Знайти всі трійки цілих чисел (a, b, c), де кожне число не перевищує n, і виконується рівність a² + b² = c².

Алгоритм: Перебір усіх можливих комбінацій a, b, c у межах [1, n]. Для уникнення дублікатів b починається з a, а c — з b.

Ключові моменти: Оптимізація шляхом обмеження діапазонів, збереження унікальних трійок.

---

2. Трикутник Паскаля

Завдання: Побудувати перші n рядків трикутника, де кожен елемент — сума двох верхніх.

Алгоритм: Створення списку списків. Початковий рядок містить 1. Внутрішні елементи обчислюються з попереднього рядка.

Ключові моменти: Використання центрування для візуалізації трикутника.

---

3. Решето Ератосфена

Завдання: Знайти прості числа серед перших 1000 чисел.

Алгоритм: Викреслення кратних простих чисел, починаючи з 2.

Ключові моменти: Використання булевого масиву для позначення простих чисел.

---

4. Розклад на прості множники

Завдання: Розкласти задане число на добуток простих множників.

Алгоритм: Послідовне ділення на 2, потім на непарні числа до √n. Залишок >2 додається як простий.

Ключові моменти: Ефективний перебір дільників.

---

5. Паліндроми-квадрати

Завдання: Знайти числа до 100, які є паліндромами і залишаються ними при квадратуванні.

Алгоритм: Перевірка паліндромності через обертання рядка.

Ключові моменти: Обмеження перебору першими 100 числами.

---

6. Число словами

Завдання: Перетворити число ≤1000 на текст українською мовою.

Алгоритм: Використання масивів для одиниць, десятків, сотень. Обробка спеціальних випадків (наприклад, 1000).

Ключові моменти: Капіталізація результату.

---

7. Близнюцькі прості числа

Завдання: Знайти пари простих чисел у проміжку [n, 2n], різниця яких дорівнює 2.

Алгоритм: Використання решета для знаходження простих чисел. Пошук сусідніх пар.

Ключові моменти: Оптимізація через обмеження до 2n.

---

8. Форматування тексту

Завдання: Відредагувати текст так, щоб ширина рядка не перевищувала n символів, зберігаючи абзаци.

Алгоритм: Розбиття на слова, формування рядків з урахуванням максимальної довжини.

Ключові моменти: Обробка абзаців через подвійні переводи рядків.

---

9. Стеганографія

Завдання: Приховати повідомлення у тексті через кількість пробілів між словами (0 — 1 пробіл, 1 — 2 пробіли).

Алгоритм: Кодування ASCII символів у бінарний вигляд, зміна пробілів. Декодування через аналіз пробілів.

Ключові моменти: Використання регулярних виразів для розбиття тексту.

---

10. Чарівні вектори

Завдання: Знайти послідовності довжини N, де сума елементів дорівнює їх добутку.

Алгоритм: Генерація всіх можливих комбінацій чисел від 1 до N. Перевірка умови.

Ключові моменти: Використання itertools для комбінацій з повтореннями.

---

11. Оптимальна столиця

Завдання: Знайти координати міста, що мінімізує суму манхеттенських відстаней до інших міст.

Алгоритм: Медіана координат x та y дає оптимальне рішення.

Ключові моменти: Сортування координат і вибір середнього елемента.  
**Код:**

(1)

def find\_pythagorean\_triples(n):

triples = []

# Перебираємо всі можливі значення для a, b, c

for a in range(1, n+1):

for b in range(a, n+1): # b починається з a, щоб уникнути повторень (a, b) і (b, a)

for c in range(b, n+1): # c починається з b, щоб уникнути повторень

if a\*\*2 + b\*\*2 == c\*\*2:

triples.append((a, b, c))

return triples

# Введення числа n

n = int(input("Введіть число n: "))

# Знаходимо всі пітівські трійки

result = find\_pythagorean\_triples(n)

# Виводимо результат

if result:

print("Знайдені пітівські трійки:")

for triple in result:

print(triple)

else:

print("Пітівські трійки не знайдено.")

(2)

def pascal\_triangle(n):

triangle = [] # Список для зберігання трикутника

# Створення кожного ряду трикутника

for i in range(n):

row = [1] \* (i + 1) # Початковий рядок з одиниць

# Заповнення внутрішніх елементів

for j in range(1, i):

row[j] = triangle[i - 1][j - 1] + triangle[i - 1][j]

triangle.append(row) # Додаємо рядок до трикутника

# Виведення трикутника

for row in triangle:

print(" ".join(map(str, row)).center(n \* 2)) # Виведення рядка з центруванням

# Введення числа n

n = int(input("Введіть кількість рядків трикутника Паскаля: "))

# Побудова і виведення трикутника

pascal\_triangle(n)

(3)

def sieve(n):

primes = [True] \* (n+1)

primes[0] = primes[1] = False

for i in range(2, int(n\*\*0.5) + 1):

if primes[i]:

for j in range(i\*i, n+1, i):

primes[j] = False

return [i for i, is\_prime in enumerate(primes) if is\_prime]

print(sieve(1000))

(4)

def prime\_factors(n):

factors = []

# Перевіряємо на простий множник 2

while n % 2 == 0:

factors.append(2)

n //= 2

# Перевіряємо на прості множники непарні числа від 3 до √n

for i in range(3, int(n\*\*0.5) + 1, 2):

while n % i == 0:

factors.append(i)

n //= i

# Якщо після всіх перевірок n більше 2, то це просте число

if n > 2:

factors.append(n)

return factors

# Введення числа

n = int(input("Введіть число для розкладу на прості множники: "))

# Розкладаємо на прості множники

factors = prime\_factors(n)

# Виводимо результат

print(f"{n} = {', '.join(map(str, factors))}")

(5)

def is\_palindrome(number):

"""Перевірка, чи є число паліндромом."""

s = str(number)

return s == s[::-1]

def find\_palindromic\_squares():

result = []

# Перебираємо числа від 1 до 100

for num in range(1, 101):

if is\_palindrome(num):

square = num \*\* 2

if is\_palindrome(square):

result.append((num, square))

return result

# Отримуємо результати

palindromic\_numbers = find\_palindromic\_squares()

# Виводимо знайдені числа та їх квадрати

print("Знайдені паліндроми (число, квадрат):")

for num, square in palindromic\_numbers:

print(f"{num}^2 = {square}")

(6)

def number\_to\_words(n):

# Массиви для основних чисел

ones = ["", "один", "два", "три", "чотири", "п’ять", "шість", "сім", "вісім", "дев’ять", "десять",

"одинадцять", "дванадцять", "тринадцять", "чотирнадцять", "п’ятнадцять", "шістнадцять",

"сімнадцять", "вісімнадцять", "дев’ятнадцять"]

tens = ["", "", "двадцять", "тридцять", "сорок", "п’ятдесят", "шістдесят", "сімдесят", "вісімдесят", "дев’яносто"]

hundreds = ["", "сто", "двісті", "триста", "чотириста", "п’ятсот", "шістсот", "сімсот", "вісімсот", "дев’ятсот"]

if n == 1000:

return "тисяча"

words = []

# Перевіряємо сотні

if n >= 100:

h = n // 100

words.append(hundreds[h])

n %= 100

# Перевіряємо десятки та одиниці

if n >= 20:

t = n // 10

words.append(tens[t])

n %= 10

if 0 < n < 20:

words.append(ones[n])

elif n == 0 and not words:

words.append("нуль")

return " ".join(words).capitalize()

# Тестування

n = int(input("Введіть число (n ≤ 1000): "))

print(number\_to\_words(n))

(7)

def sieve\_of\_eratosthenes(limit):

"""Повертає список булевих значень для чисел від 0 до limit,

де True означає, що число є простим."""

sieve = [True] \* (limit + 1)

sieve[0] = sieve[1] = False # 0 та 1 не є простими

for i in range(2, int(limit\*\*0.5) + 1):

if sieve[i]:

for j in range(i\*i, limit + 1, i):

sieve[j] = False

return sieve

def find\_twin\_primes(n):

# Створюємо решето для чисел до 2n

limit = 2 \* n

sieve = sieve\_of\_eratosthenes(limit)

twin\_primes = []

# Перебираємо прості числа між n та 2n

for i in range(n, limit - 1): # перевіряємо до 2n - 2, оскільки шукаємо пару i та i+2

if sieve[i] and i + 2 <= limit and sieve[i + 2]:

twin\_primes.append((i, i + 2))

return twin\_primes

# Введення числа n

n = int(input("Введіть число n: "))

# Знаходимо та виводимо всі пари простих чисел з різницею 2 в інтервалі [n, 2n]

pairs = find\_twin\_primes(n)

print(f"Пари простих чисел з різницею 2 в інтервалі [{n}, {2\*n}]:")

if pairs:

for a, b in pairs:

print(f"{a}, {b}")

else:

print("Не знайдено жодної пари.")

(8)

def format\_text(text, n):

# Розділяємо текст на абзаци

paragraphs = text.split('\n\n')

formatted\_text = []

for paragraph in paragraphs:

# Розбиваємо абзац на слова

words = paragraph.split()

# Поточний рядок, в який будемо додавати слова

current\_line = ""

for word in words:

# Якщо поточний рядок плюс слово не перевищує максимальну ширину, додаємо слово

if len(current\_line) + len(word) + (1 if current\_line else 0) <= n:

if current\_line:

current\_line += " " + word

else:

current\_line = word

else:

# Якщо рядок перевищує ширину, додаємо його до відформатованого тексту

formatted\_text.append(current\_line)

# Починаємо новий рядок з поточного слова

current\_line = word

# Додаємо останній сформований рядок в абзаці

if current\_line:

formatted\_text.append(current\_line)

# Додаємо два символи нового рядка для кожного абзацу

formatted\_text.append("") # Додає один абзацний розрив

# З'єднуємо всі частини і повертаємо відформатований текст

return '\n'.join(formatted\_text).strip()

# Приклад введення

text = """Це приклад тексту, який ми хочемо відредагувати так, щоб його ширина була не більше n символів.

Ми розіб'ємо його на слова і перенесемо ці слова в нові рядки, зберігаючи абзаци.

Наприклад, для ширини 50 символів, кожен рядок не перевищуватиме цієї довжини."""

n = 50 # Максимальна ширина рядка

# Виводимо відредагований текст

formatted\_text = format\_text(text, n)

print(formatted\_text)

(9)

import re

import sys

import textwrap

def encode\_message(cover\_text, secret\_message):

"""

Зашифровує секретне повідомлення в тексті.

cover\_text: рядок – текст‑«обкладинка».

secret\_message: рядок – повідомлення, яке потрібно приховати.

Повертає новий текст, де між першими gap\_count = 8\*len(secret\_message)

парами слів пробіл(и) замінені відповідно до бітів повідомлення:

- якщо біт = '0' → 1 пробіл;

- якщо біт = '1' → 2 пробіли.

Інші розриви між словами залишаються по 1 пробілу.

"""

# Розбиваємо текст на слова (при цьому оригінальні пробіли не зберігаються)

words = cover\_text.split()

gaps\_total = len(words) - 1

bits\_needed = len(secret\_message) \* 8

if gaps\_total < bits\_needed:

raise ValueError(f"Не вистачає слів для зашифрування повідомлення. "

f"Потрібно хоча б {bits\_needed + 1} слів, а знайдено {len(words)}.")

# Формуємо рядок бітів для секретного повідомлення (8 біт на символ)

bits = ""

for ch in secret\_message:

bits += format(ord(ch), '08b') # наприклад, 'A' -> '01000001'

# Формуємо новий текст, змінюючи тільки перші bits\_needed розривів між словами

encoded\_text = ""

gap\_index = 0

for i, word in enumerate(words):

encoded\_text += word

# Якщо не останнє слово, додаємо пробільний розрив

if i < len(words) - 1:

if gap\_index < bits\_needed:

# Якщо біт = '0' – один пробіл, інакше (біт '1') – два пробіли

if bits[gap\_index] == '0':

encoded\_text += " "

else:

encoded\_text += " "

gap\_index += 1

else:

# Для решти розривів стандартно – один пробіл

encoded\_text += " "

return encoded\_text

def decode\_message(stego\_text, num\_chars):

"""

Розшифровує секретне повідомлення з тексту, в якому воно було зашифроване.

stego\_text: текст, у якому закодовано повідомлення.

num\_chars: кількість символів секретного повідомлення.

Функція аналізує пробільні розриви між словами. Перші 8\*num\_chars розривів

інтерпретуються: 1 пробіл → '0', 2 (або більше) пробілів → '1'.

Потім кожні 8 біт перетворюються у символ згідно ASCII.

"""

# За допомогою регулярного виразу отримуємо послідовність: слово, пробіли, слово, пробіли, ...

tokens = re.split(r'(\s+)', stego\_text)

# Отримуємо всі пробільні розриви, що знаходяться між словами (вони розташовані зазвичай на непарних позиціях)

gaps = tokens[1::2]

bits = ""

# Аналізуємо лише перші num\_chars\*8 розривів (якщо їх менше, то повідомлення не повне)

for gap in gaps[:num\_chars \* 8]:

# Якщо кількість пробілів рівна 1, інтерпретуємо як '0', якщо 2 або більше – як '1'

if gap.count(" ") == 1:

bits += "0"

else:

bits += "1"

secret\_message = ""

# Розбиваємо рядок бітів на блоки по 8 біт і перетворюємо їх у символи

for i in range(0, len(bits), 8):

byte = bits[i:i+8]

if len(byte) < 8:

break

secret\_message += chr(int(byte, 2))

return secret\_message

# --- Демонстрація роботи програми ---

if \_\_name\_\_ == '\_\_main\_\_':

# Приклад: зчитування тексту-«обкладинки» (можна ввести з консолі або задати рядок)

cover\_text = ("Це приклад тексту, в якому буде приховано секретне повідомлення. "

"Ми використовуємо простий метод стеганографії, що базується на розміщенні "

"додаткових пробілів між словами. Цей текст має достатню кількість слів, "

"щоб можна було зашифрувати декілька символів. Спробуйте знайти приховане повідомлення!")

# Введення секретного повідомлення (наприклад, "Hello")

secret\_message = input("Введіть секретне повідомлення для зашифрування: ")

try:

# Зашифровуємо повідомлення в тексті

stego\_text = encode\_message(cover\_text, secret\_message)

print("\nТекст з прихованим повідомленням:")

print(textwrap.fill(stego\_text, width=80))

except ValueError as ve:

print("Помилка:", ve)

sys.exit(1)

# Для демонстрації розшифрування просимо ввести кількість символів повідомлення,

# або можемо використати довжину secret\_message, якщо це відомо

num\_chars = len(secret\_message)

decoded\_message = decode\_message(stego\_text, num\_chars)

print("\nРозшифроване повідомлення:")

print(decoded\_message)

(10)

import itertools

def is\_magic\_vector(vector):

"""

Перевіряє, чи є вектор чарівним:

- Сума елементів повинна дорівнювати їх добутку.

"""

return sum(vector) == prod(vector)

def prod(lst):

"""

Обчислює добуток елементів списку.

"""

result = 1

for num in lst:

result \*= num

return result

def find\_magic\_vectors(n):

"""

Знаходить всі чарівні вектори довжиною n.

"""

# Генеруємо всі можливі варіанти чисел від 1 до n

candidates = list(itertools.combinations\_with\_replacement(range(1, n + 1), n))

magic\_vectors = []

for candidate in candidates:

if is\_magic\_vector(candidate):

magic\_vectors.append(candidate)

return magic\_vectors

# Введення числа N

N = int(input("Введіть N (довжина чарівного вектора): "))

# Знайдемо всі чарівні вектори для заданого N

magic\_vectors = find\_magic\_vectors(N)

print(f"Чарівні вектори для N = {N}:")

for vector in magic\_vectors:

print(vector)

(11)

def find\_capital(cities):

"""

cities - список кортежів (x, y) координат міст.

Функція повертає координати столиці, що мінімізують суму відстаней.

"""

n = len(cities)

# Витягуємо окремо координати x та y

xs = sorted([x for x, y in cities])

ys = sorted([y for x, y in cities])

# Для непарної кількості міст медіана – це елемент посередині.

# Для парної кількості міст будь-яке значення між двома центральними елементами дає мінімум.

# Для простоти вибираємо один із них (наприклад, лівий).

if n % 2 == 1:

med\_x = xs[n // 2]

med\_y = ys[n // 2]

else:

med\_x = xs[n // 2 - 1]

med\_y = ys[n // 2 - 1]

return med\_x, med\_y

if \_\_name\_\_ == '\_\_main\_\_':

# Зчитування даних. Припустимо, спочатку вводиться кількість міст, а далі N рядків з координатами.

try:

N = int(input("Введіть кількість міст N: "))

except ValueError:

print("Будь ласка, введіть ціле число.")

exit(1)

cities = []

print("Введіть координати міст (x y), по одному місту в рядку:")

for i in range(N):

try:

x, y = map(int, input(f"Місто {i+1}: ").split())

cities.append((x, y))

except ValueError:

print("Будь ласка, введіть два цілі числа, розділених пробілом.")

exit(1)

capital = find\_capital(cities)

print(f"\nОптимальні координати столиці: {capital[0]}, {capital[1]}")

## **5.3 Завдання 3**

РВведення та оброблення переліку книг у бібліотеці

**Пояснення:**

Ця програма реалізує функціональність для обробки переліку книг у бібліотеці. Користувач може виконувати такі дії:

1. **Додавання книги** — нові книги можна додавати до бібліотеки, вказуючи назву, автора, рік видання та ISBN.
2. **Відображення всіх книг** — список всіх книг у бібліотеці виводиться на екран.
3. **Пошук книги за автором** — за допомогою цього функціоналу можна знайти всі книги певного автора, вказавши його ім'я.
4. **Пошук книги за назвою** — дає змогу знайти книги за назвою.
5. **Видалення книги** — можна видалити книгу зі списку, вибравши її за індексом.
6. **Вихід з програми** — завершення роботи програми.

Ця програма допомагає організувати каталог бібліотеки з можливістю пошуку та редагування даних про книги.

**Код:**

\*\*Після запуску треба додати 2-3 книги а так все працює я перевіряла\*\*

def print\_menu():

print("\nМеню бібліотеки:")

print("1. Додати книгу")

print("2. Відобразити всі книги")

print("3. Знайти книгу за автором")

print("4. Знайти книгу за назвою")

print("5. Видалити книгу")

print("6. Вийти")

def add\_book(book\_list):

title = input("Введіть назву книги: ")

author = input("Введіть автора книги: ")

year = input("Введіть рік видання: ")

isbn = input("Введіть ISBN книги: ")

book = {"title": title, "author": author, "year": year, "isbn": isbn}

book\_list.append(book)

print("Книгу успішно додано до бібліотеки!")

def display\_books(book\_list):

if not book\_list:

print("Бібліотека порожня!")

return

print("\nСписок книг у бібліотеці:")

for index, book in enumerate(book\_list, 1):

print(f"{index}. Назва: {book['title']}, Автор: {book['author']}, Рік: {book['year']}, ISBN: {book['isbn']}")

def search\_by\_author(book\_list):

search = input("Введіть ім'я автора для пошуку: ").lower()

results = [book for book in book\_list if search in book['author'].lower()]

if results:

print("\nЗнайдені книги:")

for book in results:

print(f"Назва: {book['title']}, Автор: {book['author']}, Рік: {book['year']}, ISBN: {book['isbn']}")

else:

print("Книг від цього автора не знайдено.")

def search\_by\_title(book\_list):

search = input("Введіть назву книги для пошуку: ").lower()

results = [book for book in book\_list if search in book['title'].lower()]

if results:

print("\nЗнайдені книги:")

for book in results:

print(f"Назва: {book['title']}, Автор: {book['author']}, Рік: {book['year']}, ISBN: {book['isbn']}")

else:

print("Книг з цією назвою не знайдено.")

def delete\_book(book\_list):

display\_books(book\_list)

if not book\_list:

return

try:

index = int(input("Введіть номер книги, яку потрібно видалити: "))

if 1 <= index <= len(book\_list):

removed\_book = book\_list.pop(index - 1)

print(f"Книгу '{removed\_book['title']}' успішно видалено з бібліотеки!")

else:

print("Невірний номер книги!")

except ValueError:

print("Будь ласка, введіть коректне число.")

def main():

book\_list = []

while True:

print\_menu()

choice = input("Виберіть опцію: ")

if choice == '1':

add\_book(book\_list)

elif choice == '2':

display\_books(book\_list)

elif choice == '3':

search\_by\_author(book\_list)

elif choice == '4':

search\_by\_title(book\_list)

elif choice == '5':

delete\_book(book\_list)

elif choice == '6':

print("Вихід з програми...")

break

else:

print("Невірний вибір! Спробуйте ще раз.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

# **6 Висновки**

**Підсумки практики з програмування на Python: від основ до майстерності**

Протягом практики я пройшла значний шлях від знайомства з Python до впевненого використання його можливостей у реальних проектах.

На початковому етапі я ознайомилася з історією Python, його ключовими характеристиками та широким спектром застосувань. Мова вразила мене своєю гнучкістю, універсальністю та простотою синтаксису, що робить її ідеальним вибором для програмування на будь-якому рівні.

Опановуючи базові принципи синтаксису, я перейшла до вивчення різноманітних типів даних та структур, таких як списки, кортежі, множини та словники. Ці знання стали основою для подальшої роботи з даними та дозволили ефективно вирішувати різноманітні завдання.

Наступним етапом було освоєння алгоритмізації та побудови програм. Я навчилася структурувати код за допомогою умовних операторів, циклів та логічних конструкцій, що дало змогу створювати ефективні та зручні для підтримки програми.

Особливу увагу я приділила обробці винятків та магічним методам, що дозволило мені розробляти більш стабільні, гнучкі та надійні програми, що легко адаптуються до різних умов.

Загалом, практика стала чудовим стартом у моєму професійному розвитку в сфері програмування. Я здобула не тільки глибокі теоретичні знання, але й набуток практичних навичок, що дозволяють мені вирішувати завдання будь-якої складності — від аналізу даних до створення веб-додатків, використовуючи Python як потужний інструмент для реалізації своїх ідей.

# **ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ІНФОРМАЦІЇ**

1. **Офіційна документація Python** -<https://docs.python.org/>
2. **Python.org** -<https://www.python.org/>
3. **Ресурси на W3Schools** - https://www.w3schools.com/python/
4. **Learn Python** -<https://www.learnpython.org/>
5. **Real Python** -<https://realpython.com/>
6. **Python для початківців** -<https://python-course.eu/>
7. **Stack Overflow (Python)** -<https://stackoverflow.com/questions/tagged/python>
8. **Книга "Automate the Boring Stuff with Python"** -<https://automatetheboringstuff.com/>
9. **Курс на Codecademy** - https://www.codecademy.com/learn/learn-python-3
10. **Python Tutor** (інструмент для візуалізації коду) -<http://pythontutor.com/>