Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Лабораторна роботи №9

з дисципліни «Спеціалізовані мови програмування»

на тему

«Створення та рефакторінг програмно-інформаційного продукту засобами Python»

Виконав:

Бабич О. Ю.

Перевірив:

Щербак С. С.

Львів 2024

**Мета:** розробка програмно-інформаційного продукту засобами.

**План роботи**

Завдання 1. Створити скрипт запуску лабораторних робіт 1-8 (Runner) з єдиним меню для управління додатками використовуючи патерн FACADE https://refactoring.guru/uk/design-patterns/facade

Завдання 2. Зробити рефакторінг додатків, які були зроблені в лб 1-8, для підтримки можливості запуску через Runner

Завдання 3. Зробити рефакторинг додатків, які були зроблені в лб 1-8, використовуючи багаторівневу архітектуру додатків (див. приклад нижче) та всі принципи об’єктно-орієнтованого підходу

Завдання 4. Створити бібліотеку класів, які повторно використовуються у всіх лабораторних роботах та зробити рефакторінг додатків для підтримки цієї бібліотеки. Таких класів в бібліотеці має буде як найменш 5

Завдання 5. Додати логування функцій в класи бібліотеки програмного продукту використовуючи https://docs.python.org/uk/3/howto/logging.html

Завдання 6. Додати коментарі до програмного коду та сформувати документацію програмного продукту засобами pydoc. Документація має бути представлена у вигляді сторінок тексту на консолі, подана у веб-браузері та збережена у файлах HTML

Завдання 7. Документація та код програмного продукту має бути розміщено в GIT repo

Завдання 8. Проведіть статичний аналіз коду продукту засобами PYLINT https://pylint.readthedocs.io/en/stable/ та виправте помилки, які були ідентифіковані. Первинний репорт з помилками додайте до звіту лабораторної роботи

Завдання 9. Підготуйте звіт до лабораторной роботи

**Реалізація:**

**ЛР 1 – Введення в Python:**

**Мета:** створення консольної програми-калькулятора за допомогою основних синтаксичних конструкцій Python, з іншим завданням на заміну тестуванню та валідації.

**План роботи**

Завдання 1: Введення користувача

Створіть Python-програму, яка приймає введення користувача для двох чисел і оператора (наприклад, +, -, \*, /).

Завдання 2: Перевірка оператора

Перевірте чи введений оператор є дійсним (тобто одним із +, -, \*, /). Якщо ні, відобразіть повідомлення про помилку і попросіть користувача ввести дійсний оператор.

Завдання 3: Обчислення

Виконайте обчислення на основі введення користувача (наприклад, додавання, віднімання, множення, ділення) і відобразіть результат.

Завдання 4: Повторення обчислень

Запитайте користувача, чи він хоче виконати ще одне обчислення. Якщо так, дозвольте йому ввести нові числа і оператор. Якщо ні, вийдіть з програми.

Завдання 5: Обробка помилок

Реалізуйте обробку помилок для обробки ділення на нуль або інших потенційних помилок. Відобразіть відповідне повідомлення про помилку, якщо виникає помилка.

Завдання 6: Десяткові числа

Змініть калькулятор так, щоб він обробляв десяткові числа (плаваючу кому) для більш точних обчислень.

Завдання 7: Додаткові операції

Додайте підтримку додаткових операцій, таких як піднесення до степеня (^), квадратний корінь (√) і залишок від ділення (%).

Завдання 8: Функція пам'яті

Реалізуйте функцію пам'яті, яка дозволяє користувачам зберігати і відновлювати результати. Додайте можливості для зберігання та отримання значень з пам'яті.

Завдання 9: Історія обчислень

Створіть журнал, який зберігає історію попередніх обчислень, включаючи вираз і результат. Дозвольте користувачам переглядати історію своїх обчислень.

Завдання 10: Налаштування користувача

Надайте користувачам можливість налаштувати поведінку калькулятора, таку як зміну кількості десяткових розрядів, які відображаються, або налаштування функцій пам'яті.  
  
**Реалізація:**

**Папка AppSettings, файл AppSettings.py:**  
decimal\_places = 2

**Папка functions, файл functions.py:**import math

from tools.tools import get\_from\_memory

def get\_numbers():

choice = input("Використати значення з пам'яті? (так/ні): ").lower()

if choice == 'так':

num1 = get\_from\_memory()

if num1 is None:

return get\_numbers()

print(f"Використано значення з пам'яті: {num1}")

else:

try:

num1 = float(input("Введіть перше число: "))

except ValueError:

print("Помилка: потрібно ввести дійсне число.")

return get\_numbers()

try:

num2 = float(input("Введіть друге число: "))

return num1, num2

except ValueError:

print("Помилка: потрібно ввести дійсне число.")

return get\_numbers()

def get\_operator():

operator = input("Введіть оператор (+, -, \*, /, %, ^, √): ")

if operator in ['+', '-', '\*', '/', '%', '^', '√']:

return operator

else:

print("Помилка: некоректний оператор.")

return get\_operator()

def calculate(num1, num2, operator):

try:

if operator == '+':

return num1 + num2

elif operator == '-':

return num1 - num2

elif operator == '\*':

return num1 \* num2

elif operator == '/':

if num2 == 0:

raise ZeroDivisionError

return num1 / num2

elif operator == '%':

return num1 % num2

elif operator == '^':

return num1 \*\* num2

elif operator == '√':

if num1 < 0:

raise ValueError("Корінь з від'ємного числа не існує.")

return math.sqrt(num1)

except ZeroDivisionError:

print("Помилка: ділення на нуль.")

except ValueError as e:

print(f"Помилка: {e}")

**Папка GlobalVariables, файл GlobalVariables.py:**

memory = None

history = []

decimal\_places = 2

**Папка interface, файл interface.py:**

def get\_user\_input(prompt):

return input(prompt)

def display\_output(message):

print(message)

**Папка logs, файл logs.py:**

def log\_error(error\_message):

with open("error\_log.txt", "a") as log\_file:

log\_file.write(f"Помилка: {error\_message}\n")

def log\_calculation(expression, result):

with open("calculation\_log.txt", "a") as log\_file:

log\_file.write(f"Обчислення: {expression} = {result}\n")

**Папка tools, файл tools.py:**

from GlobalVariables.GlobalVariables import memory, history

def save\_to\_memory(result):

global memory

memory = result

print(f"Збережено у пам'яті: {memory}")

def get\_from\_memory():

if memory is None:

print("Помилка: пам'ять порожня.")

return None

return memory

def show\_history():

for i, entry in enumerate(history, 1):

print(f"{i}: {entry}")

def customize():

global decimal\_places

try:

decimal\_places = int(input("Введіть кількість десяткових розрядів для результатів (за замовчуванням 2): "))

except ValueError:

print("Помилка: потрібно ввести ціле число.")

customize()

**Файл main.py:**

from functions.functions import get\_operator, get\_numbers, calculate

from GlobalVariables.GlobalVariables import memory, history, decimal\_places

from tools.tools import save\_to\_memory, get\_from\_memory, show\_history, customize

def run\_calculator():

global memory, history, decimal\_places

while True:

operator = get\_operator()

if operator == '√':

choice = input("Використати значення з пам'яті? (так/ні): ").lower()

if choice == 'так':

num1 = get\_from\_memory()

if num1 is None:

continue

else:

num1 = float(input("Введіть число для операції √: "))

num2 = 0

else:

num1, num2 = get\_numbers()

result = calculate(num1, num2, operator)

if result is not None:

result = round(result, decimal\_places)

print(f"Результат: {result}")

history.append(f"{num1} {operator} {num2} = {result}" if operator != '√' else f"√{num1} = {result}")

save\_choice = input("Зберегти результат у пам'ять? (так/ні): ").lower()

if save\_choice == 'так':

save\_to\_memory(result)

choice = input("Чи хочете виконати ще одне обчислення? (так/ні): ").lower()

if choice != 'так':

break

history\_choice = input("Бажаєте переглянути історію? (так/ні): ").lower()

if history\_choice == 'так':

show\_history()

customize\_choice = input("Бажаєте налаштувати калькулятор? (так/ні): ").lower()

if customize\_choice == 'так':

customize()

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

run\_calculator()

**ЛР 2 Основи побудови об’єктно-орієнтованих додатків на Python:**

**Мета:** Розробка консольного калькулятора в об’єктно орієнтованому стилі з використанням класів

**План роботи**

Завдання 1: Створення класу Calculator

Створіть клас Calculator, який буде служити основою для додатка калькулятора.

Завдання 2: Ініціалізація калькулятора

Реалізуйте метод \_\_init\_\_ у класі Calculator для ініціалізації необхідних атрибутів або змінних.

Завдання 3: Введення користувача

Перемістіть функціональність введення користувача в метод у межах класу Calculator. Метод повинен приймати введення для двох чисел і оператора.

Завдання 4: Перевірка оператора

Реалізуйте метод у класі Calculator, щоб перевірити, чи введений оператор є дійсним (тобто одним із +, -, \*, /). Відобразіть повідомлення про помилку, якщо він не є дійсним.

Завдання 5: Обчислення

Створіть метод у класі Calculator, який виконує обчислення на основі введення користувача (наприклад, додавання, віднімання, множення, ділення).

Завдання 6: Обробка помилок

Реалізуйте обробку помилок у межах класу Calculator для обробки ділення на нуль або інших потенційних помилок. Відобразіть відповідні повідомлення про помилку.

Завдання 7: Повторення обчислень

Додайте метод до класу Calculator, щоб запитати користувача, чи він хоче виконати ще одне обчислення. Якщо так, дозвольте йому ввести нові числа і оператор. Якщо ні, вийдіть з програми.

Завдання 8: Десяткові числа

Модифікуйте клас Calculator для обробки десяткових чисел (плаваюча кома) для більш точних обчислень.

Завдання 9: Додаткові операції

Розширте клас Calculator, щоб підтримувати додаткові операції, такі як піднесення до степеня (^), квадратний корінь (√) та залишок від ділення (%).

Завдання 10: Інтерфейс, зрозумілий для користувача

Покращте інтерфейс користувача у межах класу Calculator, надавши чіткі запити, повідомлення та форматування виводу для зручності читання.

**Реалізація:**

**Папка AppSettings, файл AppSettings.py:**  
decimal\_places = 2

**Папка classes, файл calculator.py:**

from GlobalVariables.GlobalVariables import memory, history

from functions.functions import get\_operator, calculate

from tools.tools import save\_to\_memory, get\_from\_memory, show\_history

from AppSettings.AppSettings import decimal\_places

class Calculator:

def \_\_init\_\_(self):

self.memory = memory

self.history = history

self.decimal\_places = decimal\_places

def get\_numbers(self):

choice = input("Використати значення з пам'яті? (так/ні): ").lower()

if choice not in ['так', 'ні']:

print("Помилка: введіть так або ні.")

return self.get\_numbers()

if choice == 'так':

num1 = get\_from\_memory()

if num1 is None:

return self.get\_numbers()

print(f"Використано значення з пам'яті: {num1}")

else:

try:

num1 = float(input("Введіть перше число: "))

except ValueError:

print("Помилка: потрібно ввести дійсне число.")

return self.get\_numbers()

try:

num2 = float(input("Введіть друге число: "))

return num1, num2

except ValueError:

print("Помилка: потрібно ввести дійсне число.")

return self.get\_numbers()

def run(self):

while True:

num1, num2 = self.get\_numbers()

operator = get\_operator()

result = calculate(num1, num2, operator)

if result is not None:

result = round(result, self.decimal\_places)

print(f"Результат: {result}")

self.history.append(f"{num1} {operator} {num2} = {result}")

self.ask\_save\_to\_memory(result)

if not self.ask\_continue():

break

if self.ask\_view\_history():

show\_history()

if self.ask\_customize():

self.customize()

def ask\_save\_to\_memory(self, result):

save\_choice = input("Зберегти результат у пам'ять? (так/ні): ").lower()

if save\_choice == 'так':

save\_to\_memory(result)

def ask\_continue(self):

return input("Чи хочете виконати ще одне обчислення? (так/ні): ").lower() == 'так'

def ask\_view\_history(self):

return input("Бажаєте переглянути історію? (так/ні): ").lower() == 'так'

def ask\_customize(self):

return input("Бажаєте налаштувати калькулятор? (так/ні): ").lower() == 'так'

def customize(self):

try:

new\_decimal\_places = int(input("Введіть кількість десяткових розрядів для результатів (за замовчуванням 2): "))

self.decimal\_places = new\_decimal\_places

print(f"Кількість десяткових розрядів змінена на: {self.decimal\_places}")

self.history.append(f"Користувач змінив кількість десяткових знаків на: {self.decimal\_places}")

except ValueError:

print("Помилка: потрібно ввести ціле число.")

self.history.append("Помилка при налаштуванні десяткових розрядів.")

**Папка functions, файл functions.py:**import math

def validate\_input(prompt, validation\_fn, error\_message):

while True:

user\_input = input(prompt)

if validation\_fn(user\_input):

return user\_input

print(error\_message)

def get\_operator():

return validate\_input("Введіть оператор (+, -, \*, /, %, ^, √): ",

lambda op: op in ['+', '-', '\*', '/', '%', '^', '√'],

"Помилка: некоректний оператор.")

def calculate(num1, num2, operator):

try:

operations = {

'+': lambda: num1 + num2,

'-': lambda: num1 - num2,

'\*': lambda: num1 \* num2,

'/': lambda: num1 / num2 if num2 != 0 else None,

'%': lambda: num1 % num2,

'^': lambda: num1 \*\* num2,

'√': lambda: math.sqrt(num1) if num1 >= 0 else None,

}

return operations.get(operator, lambda: None)()

except (ZeroDivisionError, ValueError) as e:

print(f"Помилка: {e}")

return None

**Папка GlobalVariables, файл GlobalVariables.py:**

memory = None

history = []

**Папка interface, файл interface.py:**

def get\_user\_input(prompt):

return input(prompt)

def display\_output(message):

print(message)

**Папка logs, файл logs.py:**

def log\_error(error\_message):

with open("error\_log.txt", "a") as log\_file:

log\_file.write(f"Помилка: {error\_message}\n")

def log\_calculation(expression, result):

with open("calculation\_log.txt", "a") as log\_file:

log\_file.write(f"Обчислення: {expression} = {result}\n")

**Папка tools, файл tools.py:**

from GlobalVariables.GlobalVariables import memory, history

def save\_to\_memory(result):

global memory

memory = result

print(f"Збережено у пам'яті: {memory}")

def get\_from\_memory():

if memory is None:

print("Помилка: пам'ять порожня.")

return None

return memory

def show\_history():

if history:

for i, entry in enumerate(history, 1):

print(f"{i}: {entry}")

else:

print("Історія порожня.")

**Файл main.py:**

import sys

import os

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

parent\_dir = os.path.abspath(os.path.join(current\_dir, '..'))

sys.path.append(parent\_dir)

from classes.calculator import Calculator

def main():

calculator = Calculator()

calculator.run()

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

main()

**ЛР 3 Розробка ASCII ART генератора для візуалізації текстових даних:**

**Мета:** створення додатка Генератора ASCII-арту

**План роботи**

Завдання 1: Введення користувача

Створіть Python-програму, яка приймає введення користувача для слова або фрази, яку треба перетворити в ASCII-арт.

Завдання 2: Бібліотека ASCII-арту

Інтегруйте бібліотеку ASCII-арту (наприклад, pyfiglet або art) у вашу програму для генерації ASCII-арту з введення користувача

Завдання 3: Вибір шрифту

Дозвольте користувачам вибирати різні стилі шрифтів для свого ASCII-арту. Надайте список доступних шрифтів та дозвольте їм вибрати один.

Завдання 4: Колір тексту

Реалізуйте опцію вибору користувачем кольору тексту для їхнього ASCII-арту. Підтримуйте основний вибір кольорів (наприклад, червоний, синій, зелений).

Завдання 5: Форматування виводу

Переконайтеся, що створений ASCII-арт правильно відформатований та вирівнюється на екрані для зручності читання.

Завдання 6: Збереження у файл

Додайте функціональність для збереження створеного ASCII-арту у текстовому файлі, щоб користувачі могли легко завантажувати та обмінюватися своїми творіннями.

Завдання 7: Розмір ARTу

Дозвольте користувачам вказувати розмір (ширина і висота) ASCII-арту, який вони хочуть створити. Масштабуйте текст відповідно.

Завдання 8: Вибір символів

Дозвольте користувачам вибирати символи, які вони хочуть використовувати для створення ASCII-арту (наприклад, '@', '#', '\*', тощо).

Завдання 9: Функція попереднього перегляду

Реалізуйте функцію попереднього перегляду, яка показує користувачам попередній перегляд їхнього ASCII-арту перед остаточним збереженням.

Завдання 10: Інтерфейс, зрозумілий для користувача

Створіть зручний для користувача інтерфейс командного рядку для додатка, щоб зробити його інтуїтивно зрозумілим та легким у використанні.

**Реалізація:**

**Папка AppSettings, файл AppSettings.py:**  
from classes.ASCIIArtApp import ASCIIArtApp

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

app = ASCIIArtApp()

app.run()

**Папка assets, файл assets.py:**

import pyfiglet

def get\_available\_fonts():

"""Повертає список доступних шрифтів для ASCII-арту."""

return pyfiglet.FigletFont.getFonts()

def preview\_font\_example(font):

"""Показує приклад шрифту."""

return pyfiglet.figlet\_format("Example", font=font)

**Папка classes, файл ASCIIArt.py:**

import pyfiglet

from colorama import Fore

class ASCIIArt:

def \_\_init\_\_(self, text, font="standard", color=Fore.WHITE, width=80, custom\_char="#"):

self.text = text

self.font = font

self.color = color

self.width = width

self.custom\_char = custom\_char

self.ascii\_art = ""

def generate\_art(self):

self.ascii\_art = pyfiglet.figlet\_format(self.text, font=self.font, width=self.width)

self.\_apply\_custom\_characters()

return self.color + self.ascii\_art + Fore.RESET

def \_apply\_custom\_characters(self):

self.ascii\_art = self.ascii\_art.replace('@', self.custom\_char)

self.ascii\_art = self.ascii\_art.replace('#', self.custom\_char)

def preview(self):

print("Попередній перегляд ASCII-арту:")

print(self.generate\_art())

def save\_to\_file(self, file\_path):

with open(file\_path, "w") as f:

f.write(self.ascii\_art)

print(f"ASCII-арт збережено у файл {file\_path}")

**Папка classes, файл ASCIIArtApp.py:**

from colorama import Fore

import re

from classes.ASCIIArt import ASCIIArt

from functions import functions

class ASCIIArtApp:

def \_\_init\_\_(self):

self.colors = {

1: Fore.RED,

2: Fore.BLUE,

3: Fore.GREEN,

4: Fore.WHITE,

5: Fore.YELLOW

}

def get\_font\_choice(self):

fonts = functions.get\_available\_fonts()

return functions.font\_selection(fonts)

def get\_user\_input(self):

# Завдання 1: Введення користувача

text = functions.get\_validated\_text()

# Завдання 2: Вибір шрифту

print("Доступні шрифти:")

selected\_font = self.get\_font\_choice()

# Завдання 8: Вибір символів

custom\_char = functions.get\_validated\_custom\_char()

# Завдання 7: Розмір ARTу

width = functions.get\_width()

# Завдання 4: Колір тексту

color = functions.get\_color\_choice(self.colors)

return ASCIIArt(text, selected\_font, color, width, custom\_char)

def run(self):

art = self.get\_user\_input()

# Завдання 5: Форматування виводу

print(art.generate\_art())

# Завдання 6: Збереження у файл

save\_choice = functions.get\_save\_choice()

if save\_choice == "так":

file\_path = r"C:\Users\Vlad\Desktop\унік\пітоній\Calculator\_Pro\_Max\_beta\_v033\art.txt"

art.save\_to\_file(file\_path)

# Завдання 9: Попередній перегляд

art.preview()

**Папка functions, файл functions.py:**import pyfiglet

import re

from colorama import Fore

def validate\_latin\_input(text):

"""Перевірка, чи містить введений текст тільки латинські символи."""

if re.match("^[A-Za-z0-9 ]+$", text):

return True

print("Помилка: Текст має містити тільки латинські символи та цифри.")

return False

def validate\_custom\_char(char):

"""Перевірка, чи введений символ не містить кирилицю."""

if re.match("^[^\u0400-\u04FF]+$", char): # Регулярний вираз, що виключає кирилицю

return True

print("Помилка: Символ не має містити кириличні символи.")

return False

def font\_selection(fonts):

page\_size = 10

current\_page = 0

total\_pages = len(fonts) // page\_size + (1 if len(fonts) % page\_size else 0)

while True:

print(f"\nСторінка {current\_page + 1}/{total\_pages}")

start\_index = current\_page \* page\_size

end\_index = min(start\_index + page\_size, len(fonts))

for index, font in enumerate(fonts[start\_index:end\_index], start=start\_index + 1):

print(f"{index}. {font}")

user\_action = input("\nВведіть номер шрифту (від 1 до 549), або '>' для наступної сторінки, '<' для попередньої, або 'приклад [номер]' для перегляду прикладу: ")

if user\_action == '>':

if current\_page < total\_pages - 1:

current\_page += 1

else:

print("Ви на останній сторінці.")

elif user\_action == '<':

if current\_page > 0:

current\_page -= 1

else:

print("Ви на першій сторінці.")

elif user\_action.startswith("приклад"):

try:

font\_index = int(user\_action.split()[1]) - 1

if 0 <= font\_index < len(fonts):

print(f"\nПриклад шрифту '{fonts[font\_index]}':")

print(pyfiglet.figlet\_format("Example", font=fonts[font\_index]))

else:

print("Неправильний номер шрифту.")

except (IndexError, ValueError):

print("Неправильний формат команди.")

else:

try:

font\_choice = int(user\_action) - 1

if 0 <= font\_choice < len(fonts):

return fonts[font\_choice]

else:

print("Помилка: Виберіть правильний номер шрифту.")

except ValueError:

print("Помилка: Введіть число або команду.")

def get\_available\_fonts():

return pyfiglet.FigletFont.getFonts()

def get\_validated\_text():

while True:

text = input("Введіть слово або фразу для генерації ASCII-арту (латиницею): ")

if validate\_latin\_input(text):

return text

def get\_validated\_custom\_char():

while True:

custom\_char = input("Введіть символ для ASCII-арту (або натисніть Enter для використання '#'): ")

if not custom\_char:

return '#'

if validate\_custom\_char(custom\_char):

return custom\_char

def get\_width():

default\_width = 80

while True:

width\_input = input(f"Введіть ширину ASCII-арту (за замовчуванням {default\_width}): ")

if not width\_input:

return default\_width

try:

return int(width\_input)

except ValueError:

print("Помилка: Введіть число для ширини.")

def get\_color\_choice(colors):

print("Доступні кольори:")

print("1. Червоний")

print("2. Синій")

print("3. Зелений")

print("4. Білий")

print("5. Жовтий")

while True:

try:

color\_choice = int(input("Виберіть номер кольору: "))

if 1 <= color\_choice <= 5:

return colors.get(color\_choice, Fore.WHITE)

else:

print("Помилка: Виберіть номер кольору від 1 до 5.")

except ValueError:

print("Помилка: Введіть число.")

def get\_save\_choice():

while True:

save\_choice = input("Бажаєте зберегти ASCII-арт у файл? (так/ні): ").strip().lower()

if save\_choice in ["так", "ні"]:

return save\_choice

print("Помилка: Введіть 'так' або 'ні'.")

**Папка GlobalVariables, файл GlobalVariables.py:**

GLOBAL\_DATA = {}

**Папка interface, файл interface.py:**

def display\_menu():

print("Вітаємо в Генераторі ASCII-арту!")

print("Оберіть необхідну дію:")

**Папка logs, файл logs.py:**

def log\_action(action):

with open("log.txt", "a") as log\_file:

log\_file.write(action + "\n")

**Файл main.py:**

from classes.ASCIIArtApp import ASCIIArtApp

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

app = ASCIIArtApp()

app.run()

**ЛР 4 Розробка ASCII ART генератора для візуалізації 2D-фігур:**

**Мета:** Створення Генератора ASCII-арту без використання зовнішніх бібліотек

**План роботи**

Завдання 1: Введення користувача

Створіть програму Python, яка отримує введення користувача щодо слова або фрази, яку вони хочуть перетворити в ASCII-арт.

Завдання 2: Набір символів

Визначте набір символів (наприклад, '@', '#', '\*', тощо), які будуть використовуватися для створення ASCII-арту. Ці символи будуть відображати різні відтінки.

Завдання 3: Розміри Art-у

Запитайте у користувача розміри (ширина і висота) ASCII-арту, який вони хочуть створити. Переконайтеся, що розміри в межах керованого діапазону

Завдання 4: Функція генерації Art-у

Напишіть функцію, яка генерує ASCII-арт на основі введення користувача, набору символів та розмірів. Використовуйте введення користувача, щоб визначити, які символи використовувати для кожної позиції в Art-у.

Завдання 5: Вирівнювання тексту

Реалізуйте опції вирівнювання тексту (ліво, центр, право), щоб користувачі могли вибирати, як їх ASCII-арт розміщується на екрані.

Завдання 6: Відображення мистецтва

Відобразіть створений ASCII-арт на екрані за допомогою стандартних функцій друку Python.

Завдання 7: Збереження у файл

Додайте можливість зберігати створений ASCII-арт у текстовий файл, щоб користувачі могли легко завантажувати та обмінюватися своїми творіннями.

Завдання 8: Варіанти кольорів

Дозвольте користувачам вибирати опції кольорів (чорно-білий, відтінки сірого) для свого ASCII-арту.

Завдання 9: Функція попереднього перегляду

Реалізуйте функцію попереднього перегляду, яка показує користувачам попередній перегляд їх ASCII-арту перед остаточним збереженням

Завдання 10: Інтерфейс, зрозумілий для користувача

Створіть інтерфейс для користувача у командному рядку, щоб зробити програму легкою та інтуїтивно зрозумілою для використання.

**Реалізація:**

**Папка AppSettings, файл AppSettings.py:**  
DEFAULT\_WIDTH = 80

DEFAULT\_SYMBOL = '#'

COLORS = {

1: 'Червоний',

2: 'Синій',

3: 'Зелений',

4: 'Білий',

5: 'Жовтий'

}

**Папка classes, файл ASCIIArt.py:**

class ASCIIArt:

def \_\_init\_\_(self, text, font="basic", color="Білий", width=80, custom\_char="#"):

self.text = text

self.font = font

self.color = color

self.width = width

self.custom\_char = custom\_char

self.ascii\_art = ""

self.color\_codes = {

"Червоний": "\033[91m",

"Синій": "\033[94m",

"Зелений": "\033[92m",

"Білий": "\033[97m",

"Жовтий": "\033[93m",

}

self.reset\_code = "\033[0m"

def generate\_art(self):

"""Генерує ASCII-арт за допомогою вибраного шрифту, вирівнюючи символи в одному рядку."""

lines = [""] \* 5 # Кількість рядків у символах шрифту (в даному випадку 5)

# Перетворюємо кожен символ на його ASCII представлення

for char in self.text:

ascii\_char = self.\_convert\_char\_to\_ascii(char)

ascii\_lines = ascii\_char.split("\n")

# Додаємо кожен рядок ASCII символу до відповідного рядка в загальному арті

for i in range(len(ascii\_lines)):

lines[i] += ascii\_lines[i]

self.ascii\_art = "\n".join(lines)

self.\_apply\_custom\_characters()

return self.\_apply\_color(self.ascii\_art)

def \_convert\_char\_to\_ascii(self, char):

"""Перетворює символ у його ASCII подання на основі вибраного шрифту."""

fonts = {

'basic': {

'A': [

" # ",

" # # ",

" # # ",

" ##### ",

" # # "

],

#решта букв і цифр

'block': {

'A': [

" ███ ",

" █ █ ",

" █████ ",

" █ █ ",

" █ █ "

#решта букв і цифр

],

},

'slant': {

'A': [

" /\ ",

" / \ ",

" /\_\_\_\_\ ",

" / \ ",

" / \ "

],

#решта букв і цифр

}

}

font\_dict = fonts.get(self.font, fonts['basic']) # За замовчуванням шрифт basic

return "\n".join(font\_dict.get(char.upper(), [char]))

def \_apply\_custom\_characters(self):

"""Замінює стандартні символи на користувацькі."""

self.ascii\_art = self.ascii\_art.replace('#', str(self.custom\_char)).replace('█', str(self.custom\_char))

#.replace('|', str(self.custom\_char)).replace('-', str(self.custom\_char)).replace('|', str(self.custom\_char)).replace('/', str(self.custom\_char)).replace('\_', str(self.custom\_char)).replace('\\', str(self.custom\_char))

def \_apply\_color(self, art):

"""Застосовує вибраний колір до ASCII-арту."""

color\_code = self.color\_codes.get(self.color, self.color\_codes["Білий"])

return f"{color\_code}{art}{self.reset\_code}"

def preview(self):

print("Попередній перегляд ASCII-арту:")

print(self.generate\_art())

def save\_to\_file(self, file\_path):

with open(file\_path, "w", encoding="utf-8") as f:

f.write(self.ascii\_art)

print(f"ASCII-арт збережено у файл {file\_path}")

**Папка classes, файл ASCIIArtApp.py:**

from classes.ASCIIArt import ASCIIArt

from functions import functions

class ASCIIArtApp:

def \_\_init\_\_(self):

self.colors = {

1: 'Червоний',

2: 'Синій',

3: 'Зелений',

4: 'Білий',

5: 'Жовтий'

}

def get\_font\_choice(self):

fonts = functions.get\_available\_fonts()

return functions.font\_selection(fonts)

def get\_user\_input(self):

text = functions.get\_validated\_text()

print("Доступні шрифти:")

selected\_font = self.get\_font\_choice()

custom\_char = functions.get\_validated\_custom\_char()

width = functions.get\_width()

# Вибір кольору

color = functions.get\_color\_choice(self.colors)

# Повертаємо об'єкт ASCIIArt

return ASCIIArt(text, selected\_font, color, width, custom\_char)

def run(self):

# Виклик методу для отримання введених даних

art = self.get\_user\_input()

# Генерація та відображення арту

print(art.generate\_art())

save\_choice = functions.get\_save\_choice()

if save\_choice == "так":

file\_path = r"C:\\Users\\Vlad\\Desktop\\унік\\пітоній\\ASIIART\_Pro\_Max\_beta\_v04\\art.txt"

art.save\_to\_file(file\_path)

art.preview()

**Папка functions, файл functions.py:**from AppSettings.AppSettings import COLORS

def get\_available\_fonts():

"""Повертає список доступних шрифтів для ASCII-арту."""

return ["basic", "block", "slant"]

def preview\_font\_example(font):

"""Показує приклад шрифту."""

examples = {

"basic": "Example:\n ### \n # # \n ##### \n # # \n # # \n",

"block": "Example:\n ███ \n █ █ \n █████ \n █ █ \n █ █ \n",

"slant": "Example:\n /\\ \n / \\ \n /\_\_\_\_\\ \n / \\ \n/ \\\n"

}

return examples.get(font, "Приклад не знайдено.")

def validate\_latin\_input(text):

"""Перевірка, чи містить введений текст тільки латинські символи."""

for char in text:

# Перевіряємо, чи символ є латинським символом або цифрою

if not (('a' <= char <= 'z') or ('A' <= char <= 'Z') or ('0' <= char <= '9') or char.isspace()):

print("Помилка: Текст має містити тільки латинські символи та цифри.")

return False

return True

def font\_selection(fonts):

print("Доступні шрифти:")

for i, font in enumerate(fonts):

print(f"{i + 1}. {font}")

while True:

choice = input("Виберіть шрифт: ")

try:

selected\_font = fonts[int(choice) - 1]

print(f"Приклад шрифту:\n{preview\_font\_example(selected\_font)}")

return selected\_font

except (IndexError, ValueError):

print("Помилка: Виберіть правильний номер шрифту.")

def get\_validated\_text():

while True:

text = input("Введіть слово або фразу для генерації ASCII-арту (латиницею): ")

if validate\_latin\_input(text):

return text

def validate\_custom\_char(char):

"""Перевірка, чи введений символ не містить кирилицю."""

# Тут ми можемо перевірити, чи символ не є кириличним.

if 'а' <= char <= 'я' or 'А' <= char <= 'Я':

print("Помилка: Символ не має містити кириличні символи.")

return False

return True

def get\_validated\_custom\_char():

"""Отримує і перевіряє символ для ASCII-арту."""

while True:

custom\_char = input("Введіть символ для ASCII-арту, або натисніть Enter для використання # (працює тільки з шрифтами basic та block): ")

if not custom\_char: # Якщо нічого не введено, використати '#'

return '#'

if len(custom\_char) != 1: # Перевіряємо, що введено лише один символ

print("Помилка: Ви повинні ввести рівно один символ.")

continue

if validate\_custom\_char(custom\_char): # Перевіряємо введений символ

return custom\_char

def get\_width():

"""Отримує ширину для ASCII-арту, або повертає значення за замовчуванням."""

default\_width = 80 # Значення за замовчуванням

while True:

width\_input = input(f"Введіть ширину ASCII-арту (за замовчуванням {default\_width}): ")

if not width\_input: # Якщо нічого не введено, використати значення за замовчуванням

return default\_width

try:

width = int(width\_input) # Спробуйте перетворити на число

if width > 0: # Переконайтесь, що ширина більше нуля

return width

else:

print("Помилка: Введіть число більше нуля.")

except ValueError:

print("Помилка: Введіть число для ширини.")

def get\_color\_choice(colors):

while True:

print("Виберіть колір:")

for num, color in colors.items():

print(f"{num}: {color}")

try:

color\_choice = int(input("Введіть номер кольору: "))

if color\_choice in colors:

return colors[color\_choice]

else:

print("Помилка: Виберіть номер кольору з наявних.")

except ValueError:

print("Помилка: Введіть число.")

def get\_save\_choice():

"""Запитує у користувача, чи хоче він зберегти ASCII-арт у файл."""

while True:

save\_choice = input("Бажаєте зберегти ASCII-арт у файл? (так/ні): ").strip().lower()

if save\_choice in ["так", "ні"]:

return save\_choice

print("Помилка: Введіть 'так' або 'ні'.")

**Папка interface, файл interface.py:**

def display\_menu():

print("Вітаємо в Генераторі ASCII-арту!")

print("Оберіть необхідну дію:")

**Папка logs, файл logs.py:**

def log\_action(action):

with open("log.txt", "a") as log\_file:

log\_file.write(action + "\n")

**Файл main.py:**

from classes.ASCIIArtApp import ASCIIArtApp

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

app = ASCIIArtApp()

app.run()

**ЛР 5 Розробка ASCII ART генератора для візуалізації 3D-фігур:**

**Мета:** Створення додатка для малювання 3D-фігур у ASCII-арті на основі об’єктно - орієнтованого підходу та мови Python

**План роботи**

Завдання 1: Проектування класів

Розробіть структуру класів для вашого генератора 3D ASCII-арту. Визначте основні компоненти, атрибути та методи, необхідні для програми.

Завдання 2: Введення користувача

Створіть методи у межах класу для введення користувача та вказання 3D-фігури, яку вони хочуть намалювати, та її параметрів (наприклад, розмір, кольори).

Завдання 3: Представлення фігури

Визначте структури даних у межах класу для представлення 3D-фігури. Це може включати використання списків, матриць або інших структур даних для зберігання форми фігури та її властивостей.

Завдання 4: Проектування з 3D в 2D

Реалізуйте метод, який перетворює 3D-представлення фігури у 2D-представлення, придатне для ASCII-арту.

Завдання 5: Відображення ASCII-арту

Напишіть метод у межах класу для відображення 2D-представлення 3D-фігури як ASCII-арту. Це може включати відображення кольорів і форми за допомогою символів ASCII.

Завдання 6: Інтерфейс, зрозумілий для користувача

Створіть зручний для користувача командний рядок або графічний інтерфейс користувача (GUI) за допомогою об'єктно-орієнтованих принципів, щоб дозволити користувачам спілкуватися з програмою.

Завдання 7: Маніпуляція фігурою

Реалізуйте методи для маніпулювання 3D-фігурою, такі масштабування або зміщення, щоб надавати користувачам контроль над її виглядом.

Завдання 8: Варіанти кольорів

Дозвольте користувачам вибирати варіанти кольорів для їхніх 3D ASCII-арт-фігур. Реалізуйте методи для призначення кольорів різним частинам фігури.

Завдання 9: Збереження та експорт

Додайте функціональність для зберігання згенерованого 3D ASCII-арту у текстовий файл

Завдання 10: Розширені функції

Розгляньте можливість додавання розширених функцій, таких як тінь, освітлення та ефекти перспективи, для підвищення реалізму 3D ASCII-арту.

**Реалізація:**

**Папка AppSettings, файл AppSettings.py:**  
DEFAULT\_WIDTH = 80

DEFAULT\_SYMBOL = '#'

COLORS = {

1: 'Червоний',

2: 'Синій',

3: 'Зелений',

4: 'Білий',

5: 'Жовтий'

}

**Папка classes, файл ASCIIArtApp.py:**

from functions import functions

# Словник з 2D та 3D ASCII фігурами

shapes = {

'cube': {

'3D': [

" +------+ ",

" / /| ",

" / / | ",

" +------+ + ",

" | | / ",

" | |/ ",

" +------+ "

],

'2D': [

" +------+ ",

" | | ",

" | | ",

" +------+ "

]

},

'pyramid': {

'3D': [

" /\\ ",

" / \\ ",

" /\_\_\_\_\\ ",

" /| |\\ ",

" /\_|\_\_\_\_|\_\\ "

],

'2D': [

" /\\ ",

" / \\ ",

" /\_\_\_\_\\ "

]

}

}

class ASCIIArtApp:

def \_\_init\_\_(self):

self.colors = {

1: 'Червоний',

2: 'Синій',

3: 'Зелений',

4: 'Білий',

5: 'Жовтий'

}

def get\_user\_input(self):

shape\_name = functions.get\_shape\_type() # Отримати тип фігури

# Цикл для отримання розміру фігури (2D або 3D)

while True:

dimension\_choice = input("Введіть '1' для 3D або '2' для 2D: ")

if dimension\_choice == '1':

is\_3d = True

break

elif dimension\_choice == '2':

is\_3d = False

break

else:

print("Невірний вибір. Будь ласка, спробуйте ще раз.")

color\_choice = functions.get\_color\_choice(self.colors) # Отримати вибір кольору

symbol = functions.get\_validated\_custom\_char() # Отримати символ

return shape\_name, is\_3d, color\_choice, symbol

def get\_ansi\_color(self, color):

color\_codes = {

"Червоний": "\033[91m",

"Синій": "\033[94m",

"Зелений": "\033[92m",

"Білий": "\033[97m",

"Жовтий": "\033[93m",

}

return color\_codes.get(color, "\033[97m") # За замовчуванням білий

def display\_shape(self, shape\_name, is\_3d, color, symbol):

ansi\_color = self.get\_ansi\_color(color) # Отримуємо ANSI-код кольору

reset\_color = "\033[0m" # Код для скидання кольору

shape\_type = '3D' if is\_3d else '2D'

shape = shapes[shape\_name][shape\_type]

art\_lines = [] # Список для зберігання рядків ASCII-арту

for line in shape:

# Заміна символу на вибраний символ

colored\_line = line.replace('+', symbol).replace('/', symbol).replace('\\', symbol).replace('|', symbol).replace('-', symbol).replace('\_', symbol)

art\_lines.append(f"{ansi\_color}{colored\_line}{reset\_color}") # Додаємо кольоровий рядок до списку

print(art\_lines[-1]) # Виводимо останній рядок

return art\_lines # Повертаємо список рядків ASCII-арту

def save\_to\_file(self, file\_path, shape\_name, is\_3d, symbol):

shape\_type = '3D' if is\_3d else '2D'

shape = shapes[shape\_name][shape\_type]

with open(file\_path, 'w', encoding='utf-8') as f:

for line in shape:

# Заміна символів у рядку на вибраний символ без кольору

saved\_line = line.replace('+', symbol).replace('/', symbol).replace('\\', symbol).replace('|', symbol).replace('-', symbol).replace('\_', symbol)

f.write(saved\_line + '\n') # Запис рядка без кольорових кодів

def run(self):

shape\_name, is\_3d, color\_choice, symbol = self.get\_user\_input()

# Відображення фігури

art\_lines = self.display\_shape(shape\_name, is\_3d, color\_choice, symbol)

save\_choice = functions.get\_save\_choice()

if save\_choice.lower() == "так":

file\_path = r"C:\\Users\\Vlad\\Desktop\\унік\\пітоній\\ASIIART\_Pro\_Max\_beta\_v05.1\\art.txt"

self.save\_to\_file(file\_path, shape\_name, is\_3d, symbol) # Зберігаємо файл

print(f"ASCII-арт збережено у файл {file\_path}")

**Папка functions, файл functions.py:**from AppSettings.AppSettings import COLORS

def get\_shape\_type():

shapes = ["cube", "pyramid"]

print("Доступні фігури:")

for i, shape in enumerate(shapes, 1):

print(f"{i}. {shape}")

while True:

choice = input("Виберіть фігуру: ")

try:

return shapes[int(choice) - 1]

except (IndexError, ValueError):

print("Помилка: Виберіть правильний номер фігури.")

def get\_shape\_dimension():

while True:

dimension = input("Якщо 3D - введіть 1, якщо 2D - введіть 2,: ").strip().lower()

if dimension in ['1', '2']:

return dimension

print("Неправильний ввід. Спробуйте ще раз.")

def get\_color\_choice(colors):

while True:

print("Виберіть колір:")

for num, color in colors.items():

print(f"{num}: {color}")

try:

color\_choice = int(input("Введіть номер кольору: "))

if color\_choice in colors:

return colors[color\_choice]

else:

print("Помилка: Виберіть номер кольору з наявних.")

except ValueError:

print("Помилка: Введіть число.")

def get\_validated\_custom\_char():

while True:

custom\_char = input("Введіть символ для ASCII-арту (Enter для використання #): ")

if not custom\_char:

return '#'

if len(custom\_char) == 1:

return custom\_char

print("Помилка: Ви повинні ввести рівно один символ.")

def get\_save\_choice():

while True:

save\_choice = input("Бажаєте зберегти ASCII-арт у файл? (так/ні): ").strip().lower()

if save\_choice in ["так", "ні"]:

return save\_choice

print("Помилка: Введіть 'так' або 'ні'.")

**Папка interface, файл interface.py:**

def display\_menu():

print("Вітаємо в Генераторі ASCII-арту!")

print("Оберіть необхідну дію:")

**Папка logs, файл logs.py:**

def log\_action(action):

with open("log.txt", "a") as log\_file:

log\_file.write(action + "\n")

**Файл main.py:**

from classes.ASCIIArtApp import ASCIIArtApp

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

app = ASCIIArtApp()

app.run()

**ЛР 6 Розробка та Unit тестування Python додатку:**

**Мета:** Cтворення юніт-тестів для додатка-калькулятора на основі класів

**План роботи**

Завдання 1: Тестування Додавання

Напишіть юніт-тест, щоб перевірити, що операція додавання в вашому додатку-калькуляторі працює правильно. Надайте тестові випадки як для позитивних, так і для негативних чисел.

Завдання 2: Тестування Віднімання

Створіть юніт-тести для переконання, що операція віднімання працює правильно. Тестуйте різні сценарії, включаючи випадки з від'ємними результатами.

Завдання 3: Тестування Множення

Напишіть юніт-тести, щоб перевірити правильність операції множення в вашому калькуляторі. Включіть випадки з нулем, позитивними та від'ємними числами.

Завдання 4: Тестування Ділення

Розробіть юніт-тести для підтвердження точності операції ділення. Тести повинні охоплювати ситуації, пов'язані з діленням на нуль та різними числовими значеннями.

Завдання 5: Тестування Обробки Помилок

Створіть юніт-тести, щоб перевірити, як ваш додаток-калькулятор обробляє помилки. Включіть тести для ділення на нуль та інших потенційних сценаріїв помилок. Переконайтеся, що додаток відображає відповідні повідомлення про помилки.

**Реалізація:**

**Папка AppSettings, файл AppSettings.py:**  
decimal\_places = 2

**Папка classes, файл calculator.py:**

from GlobalVariables.GlobalVariables import memory, history

from functions.functions import get\_operator, calculate

from tools.tools import save\_to\_memory, get\_from\_memory, show\_history

from AppSettings.AppSettings import decimal\_places

class Calculator:

def \_\_init\_\_(self):

self.memory = memory

self.history = history

self.decimal\_places = decimal\_places

def get\_numbers(self):

choice = input("Використати значення з пам'яті? (так/ні): ").lower()

if choice not in ['так', 'ні']:

print("Помилка: введіть так або ні.")

return self.get\_numbers()

if choice == 'так':

num1 = get\_from\_memory()

if num1 is None:

return self.get\_numbers()

print(f"Використано значення з пам'яті: {num1}")

else:

try:

num1 = float(input("Введіть перше число: "))

except ValueError:

print("Помилка: потрібно ввести дійсне число.")

return self.get\_numbers()

try:

num2 = float(input("Введіть друге число: "))

return num1, num2

except ValueError:

print("Помилка: потрібно ввести дійсне число.")

return self.get\_numbers()

def run(self):

while True:

num1, num2 = self.get\_numbers()

operator = get\_operator()

result = calculate(num1, num2, operator)

if result is not None:

result = round(result, self.decimal\_places)

print(f"Результат: {result}")

self.history.append(f"{num1} {operator} {num2} = {result}")

self.ask\_save\_to\_memory(result)

if not self.ask\_continue():

break

if self.ask\_view\_history():

show\_history()

if self.ask\_customize():

self.customize()

def ask\_save\_to\_memory(self, result):

save\_choice = input("Зберегти результат у пам'ять? (так/ні): ").lower()

if save\_choice == 'так':

save\_to\_memory(result)

def ask\_continue(self):

return input("Чи хочете виконати ще одне обчислення? (так/ні): ").lower() == 'так'

def ask\_view\_history(self):

return input("Бажаєте переглянути історію? (так/ні): ").lower() == 'так'

def ask\_customize(self):

return input("Бажаєте налаштувати калькулятор? (так/ні): ").lower() == 'так'

def customize(self):

try:

new\_decimal\_places = int(input("Введіть кількість десяткових розрядів для результатів (за замовчуванням 2): "))

self.decimal\_places = new\_decimal\_places

print(f"Кількість десяткових розрядів змінена на: {self.decimal\_places}")

self.history.append(f"Користувач змінив кількість десяткових знаків на: {self.decimal\_places}")

except ValueError:

print("Помилка: потрібно ввести ціле число.")

self.history.append("Помилка при налаштуванні десяткових розрядів.")

**Папка functions, файл functions.py:**import math

def validate\_input(prompt, validation\_fn, error\_message):

while True:

user\_input = input(prompt)

if validation\_fn(user\_input):

return user\_input

print(error\_message)

def get\_operator():

return validate\_input("Введіть оператор (+, -, \*, /, %, ^, √): ",

lambda op: op in ['+', '-', '\*', '/', '%', '^', '√'],

"Помилка: некоректний оператор.")

def calculate(num1, num2, operator):

try:

operations = {

'+': lambda: num1 + num2,

'-': lambda: num1 - num2,

'\*': lambda: num1 \* num2,

'/': lambda: num1 / num2 if num2 != 0 else None,

'%': lambda: num1 % num2,

'^': lambda: num1 \*\* num2,

'√': lambda: math.sqrt(num1) if num1 >= 0 else None,

}

return operations.get(operator, lambda: None)()

except (ZeroDivisionError, ValueError) as e:

print(f"Помилка: {e}")

return None

**Папка GlobalVariables, файл GlobalVariables.py:**

memory = None

history = []

**Папка interface, файл interface.py:**

def get\_user\_input(prompt):

return input(prompt)

def display\_output(message):

print(message)

**Папка logs, файл logs.py:**

def log\_error(error\_message):

with open("error\_log.txt", "a") as log\_file:

log\_file.write(f"Помилка: {error\_message}\n")

def log\_calculation(expression, result):

with open("calculation\_log.txt", "a") as log\_file:

log\_file.write(f"Обчислення: {expression} = {result}\n")

**Папка tools, файл tools.py:**

from GlobalVariables.GlobalVariables import memory, history

def save\_to\_memory(result):

global memory

memory = result

print(f"Збережено у пам'яті: {memory}")

def get\_from\_memory():

if memory is None:

print("Помилка: пам'ять порожня.")

return None

return memory

def show\_history():

if history:

for i, entry in enumerate(history, 1):

print(f"{i}: {entry}")

else:

print("Історія порожня.")

**Файл main.py:**

import sys

import os

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

parent\_dir = os.path.abspath(os.path.join(current\_dir, '..'))

sys.path.append(parent\_dir)

from classes.calculator import Calculator

def main():

calculator = Calculator()

calculator.run()

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

main()

**Файл test\_calculator.py**import unittest

from classes.calculator import Calculator

from functions.functions import calculate

import sys

import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

class TestCalculatorOperations(unittest.TestCase):

def setUp(self):

# Ініціалізуємо калькулятор перед кожним тестом

self.calculator = Calculator()

# Завдання 1: Тестування Додавання

def test\_addition(self):

result = calculate(5, 3, '+')

self.assertEqual(result, 8, "Додавання не працює з позитивними числами")

result = calculate(-5, -3, '+')

self.assertEqual(result, -8, "Додавання не працює з від'ємними числами")

# Завдання 2: Тестування Віднімання

def test\_subtraction(self):

result = calculate(10, 5, '-')

self.assertEqual(result, 5, "Віднімання не працює з позитивними числами")

result = calculate(5, 10, '-')

self.assertEqual(result, -5, "Віднімання не працює для отримання від'ємного результату")

result = calculate(-5, -3, '-')

self.assertEqual(result, -2, "Віднімання не працює з від'ємними числами")

# Завдання 3: Тестування Множення

def test\_multiplication(self):

result = calculate(0, 5, '\*')

self.assertEqual(result, 0, "Множення не працює з нулем")

result = calculate(3, 3, '\*')

self.assertEqual(result, 9, "Множення не працює з позитивними числами")

result = calculate(-3, 3, '\*')

self.assertEqual(result, -9, "Множення не працює з від'ємними числами")

# Завдання 4: Тестування Ділення

def test\_division(self):

result = calculate(10, 2, '/')

self.assertEqual(result, 5, "Ділення не працює з позитивними числами")

result = calculate(10, 0, '/')

self.assertIsNone(result, "Ділення на нуль не обробляється правильно")

result = calculate(-10, 2, '/')

self.assertEqual(result, -5, "Ділення не працює з від'ємними числами")

# Завдання 5: Тестування Обробки Помилок

def test\_error\_handling(self):

# Тест на неіснуючий оператор

result = calculate(10, 2, '?')

self.assertIsNone(result, "Некоректний оператор не обробляється правильно")

# Тест ділення на нуль

result = calculate(10, 0, '/')

self.assertIsNone(result, "Помилка ділення на нуль не обробляється правильно")

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

unittest.main()

**ЛР 7 Робота з API та веб-сервісами:**

**Мета:** Створення консольного об’єктно - орієнтованого додатка з використанням API та патернів проектування

**План роботи**

Завдання 1: Вибір провайдера API та патернів проектування

Виберіть надійний API, який надає через HTTP необхідні дані для віддаленого зберігання, вивантаження або реалізуйте свій. Для прикладу це може бути jsonplaceholder.org. Крім того, оберіть 2-3 патерна проектування для реалізаціі імплементаціі цієї лабораторноі роботи. Для прикладу, це може бути патерн Unit of Work та Repository

Завдання 2: Інтеграція API

Виберіть бібліотеку для роботи з API та обробки HTTP запитів (для прикладу це може бути бібліотека Requests). Інтегруйте обраний API в ваш консольний додаток на Python. Ознайомтеся з документацією API та налаштуйте необхідний API-ключ чи облікові дані.

Завдання 3: Введення користувача

Розробіть користувальницький інтерфейс, який дозволяє користувачам візуалізувати всі доступні дані в табличному вигляді та у вигляді списку. Реалізуйте механізм для збору та перевірки введеного даних користувачем.

Завдання 4: Розбір введення користувача

Створіть розбірник для видобування та інтерпретації виразів користувача на основі регулярних виразів, наприклад, для візуалізації дат, телефонів, тощо. Переконайтеся, що розбірник обробляє різні формати введення та надає зворотний зв'язок про помилки.

Завдання 5: Відображення результатів

Реалізуйте логіку для візуалізації даних через API в консолі. Обробляйте відповіді API для отримання даних у вигляді таблиць, списків. Заголовки таблиць, списків мають виділяться кольором та шрифтом, які задається користувачем

Завдання 6: Збереження даних

Реалізуйте можливості збереження даних у чіткому та читабельному форматі JSON, CSV та TXT

Завдання 7: Обробка помилок

Розробіть надійний механізм обробки помилок для керування помилками API, некоректним введенням користувача та іншими можливими проблемами. Надавайте інформативні повідомлення про помилки.

Завдання 8: Ведення історії обчислень

Включіть функцію, яка реєструє запити користувача, включаючи введені запити та відповідні результати. Дозвольте користувачам переглядати та рецензувати історію своїх запитів.

Завдання 9: Юніт-тести

Напишіть юніт-тести для перевірки функціональності вашого додатку. Тестуйте різні операції, граничні випадки та сценарії помилок.

**Реалізація:**

**Папка api\_client, файл api\_client.py:**  
import requests

class APIClient:

BASE\_URL = "https://jsonplaceholder.typicode.com"

def get\_data(self, parsed\_input):

if isinstance(parsed\_input, tuple):

resource\_name, resource\_id = parsed\_input

response = requests.get(f"{self.BASE\_URL}/{resource\_name}/{resource\_id}")

else:

resource\_name = parsed\_input

response = requests.get(f"{self.BASE\_URL}/{resource\_name}")

if response.status\_code == 404:

raise ValueError(f"Error: {response.status\_code} - Запитаний ресурс не знайдено.")

try:

data = response.json()

return data

except ValueError:

raise ValueError("Невідомий формат даних.")

**Папка app, файл app.py:**

from api\_client.api\_client import APIClient

from input\_parser.input\_parser import InputParser

from data\_visualizer.data\_visualizer import DataVisualizer

from data\_storage.data\_storage import DataStorage

from error\_handler.error\_handler import ErrorHandler

from query\_history.query\_history import QueryHistory

def main():

api\_client = APIClient()

input\_parser = InputParser()

data\_visualizer = DataVisualizer()

data\_storage = DataStorage()

error\_handler = ErrorHandler()

query\_history = QueryHistory()

resource\_counts = {

"posts": 100,

"comments": 500,

"users": 10,

"albums": 100,

"photos": 5000,

"todos": 200,

}

print("Доступні ресурси та їх кількість:")

for resource, count in resource\_counts.items():

print(f"{resource} - {count}")

while True:

try:

user\_input = input("Введіть ваш запит у форматі 'назва запиту' - якщо хочете побачити всі запити, або 'назва запиту/id запиту' - для конкретного запиту (або 'exit' для виходу): ")

if user\_input.lower() == 'exit':

break

parsed\_input = input\_parser.parse(user\_input)

data = api\_client.get\_data(parsed\_input)

data\_visualizer.visualize(data)

data\_storage.save(data)

query\_history.record(user\_input, data)

except Exception as e:

error\_handler.handle(e)

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

main()

**Папка data\_storage, файл data\_storage.py:**import json

class DataStorage:

def save(self, data):

try:

with open('data.json', 'w') as json\_file:

json.dump(data, json\_file, indent=4) # Записуємо дані у JSON форматі

except Exception as e:

print(f"Помилка при збереженні даних: {e}")

**Папка data\_visualizer, файл data\_visualizer.py:**

class DataVisualizer:

def visualize(self, data):

if isinstance(data, list):

self.display\_paginated(data) # Стратегія для відображення списків

elif isinstance(data, dict):

self.display\_item(data) # Стратегія для одного елемента

else:

print("Невідомий формат даних.")

def display\_paginated(self, data):

# Кількість записів на сторінці

items\_per\_page = 10

total\_items = len(data)

total\_pages = (total\_items + items\_per\_page - 1) // items\_per\_page # Обчислюємо загальну кількість сторінок

while True:

# Відображення кількості сторінок

print(f"\nКількість сторінок: {total\_pages}")

# Запит на введення номера сторінки

user\_input = input(f"Введіть номер сторінки (1-{total\_pages}) або 'back' для повернення: ")

if user\_input.lower() == 'back':

print("Повертаємося до основного меню...")

# Виводимо доступні запити та їх кількість

self.show\_available\_requests()

return # Повертаємося до основного меню

try:

page\_number = int(user\_input)

if page\_number < 1 or page\_number > total\_pages:

print("Помилка: Неправильний номер сторінки. Спробуйте ще раз.")

continue

# Виводимо елементи на вибраній сторінці

start\_index = (page\_number - 1) \* items\_per\_page

end\_index = min(start\_index + items\_per\_page, total\_items)

page\_items = data[start\_index:end\_index]

for item in page\_items:

self.display\_item(item)

print("\n" + "-" \* 40) # Відокремлюємо кожен елемент

except ValueError:

print("Помилка: Введіть коректний номер сторінки.")

def display\_item(self, item):

# Метод для виведення окремого елемента

print(item)

def show\_available\_requests(self):

resource\_counts = {

"posts": 100,

"comments": 500,

"users": 10,

"albums": 100,

"photos": 5000,

"todos": 200,

}

print("Доступні запити та їх кількість:")

for resource, count in resource\_counts.items():

print(f"{resource} - {count}")

**Папка error\_handler, файл error\_handler.py:**

class ErrorHandler:

def handle(self, error):

print(f"Помилка: {error}")

**Папка input\_parser, файл input\_parser.py:**

class InputParser:

def parse(self, user\_input):

if not user\_input.strip(): # Перевірка на пустий рядок

raise ValueError("Неправильний формат запиту.")

parts = user\_input.split('/')

if len(parts) == 1:

return parts[0] # Повертає тільки назву запиту

elif len(parts) == 2:

resource\_name, resource\_id = parts

if not resource\_id.isdigit(): # Перевірка, що ID є числом

raise ValueError("ID запиту має бути числом.")

return (resource\_name, resource\_id) # Повертає кортеж з назви запиту та ID

else:

raise ValueError("Неправильний формат запиту.")

**Папка query\_history, файл query\_history.py:**

class QueryHistory:

\_instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if cls.\_instance is None:

cls.\_instance = super(QueryHistory, cls).\_\_new\_\_(cls)

return cls.\_instance

def \_\_init\_\_(self):

if not hasattr(self, 'history'): # Ініціалізуємо тільки один раз

self.history = []

def record(self, query, result):

self.history.append({"query": query, "result": result})

def show\_history(self):

for entry in self.history:

print(f"Запит: {entry['query']}, Результат: {entry['result']}")

**Файл tests.py:**

import unittest

from input\_parser.input\_parser import InputParser

class TestInputParser(unittest.TestCase):

def setUp(self):

self.parser = InputParser()

def test\_parse\_invalid\_input(self):

with self.assertRaises(ValueError):

self.parser.parse("") # Тестуємо на пустий рядок

self.parser.parse("invalid\_format/") # Тестуємо на неправильний формат

self.parser.parse("posts/abc") # Тестуємо на недійсне ID (не число)

def test\_parse\_valid\_input(self):

result = self.parser.parse("posts/1")

self.assertEqual(result, ("posts", "1"))

result = self.parser.parse("comments")

self.assertEqual(result, "comments")

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

unittest.main()

**Файл main.py:**

from app.app import main

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

main()

**ЛР 8 Візуалізація та обробка даних за допомогою спеціалізованих бібліотек Python**

**Мета:** Розробка додатка для візуалізації CSV-наборів даних за допомогою Matplotlib та базових принципів ООП (наслідування, інкапсуляція, поліморфізм)

**План роботи**

Завдання 1: Вибір CSV-набору даних

Оберіть CSV-набір даних, який ви хочете візуалізувати. Переконайтеся, що він містить відповідні дані для створення змістовних візуалізацій.

Завдання 2: Завантаження даних з CSV

Напишіть код для завантаження даних з CSV-файлу в ваш додаток Python. Використовуйте бібліотеки, такі як Pandas, для спрощення обробки даних.

Завдання 3: Дослідження даних

Визначте екстремальні значення по стовцям

Завдання 4: Вибір типів візуалізацій

Визначте, які типи візуалізацій підходять для представлення вибраних наборів даних. Зазвичай це може бути лінійні графіки, стовпчикові діаграми, діаграми розсіювання, гістограми та секторні діаграми.

Завдання 5: Підготовка даних

Попередньо обробіть набір даних за необхідністю для візуалізації. Це може включати виправлення даних, фільтрацію, агрегацію або трансформацію.

Завдання 6: Базова візуалізація

Створіть базову візуалізацію набору даних, щоб переконатися, що ви можете відображати дані правильно за допомогою Matplotlib. Розпочніть з простої діаграми для візуалізації однієї змінної.

Завдання 7: Розширені візуалізації

Реалізуйте більш складні візуалізації, виходячи з характеристик набору. Поекспериментуйте з різними функціями Matplotlib та налаштуваннями.

Завдання 8: Декілька піддіаграм

Навчіться створювати кілька піддіаграм в межах одного малюнка для відображення декількох візуалізацій поруч для кращого порівняння.

Завдання 9: Експорт і обмін

Реалізуйте функціональність для експорту візуалізацій як зображень (наприклад, PNG, SVG) або інтерактивних веб-додатків (наприклад, HTML)

**Реалізація:**

**Папка classes, файл base\_visualizer.py:**

import matplotlib.pyplot as plt

from abc import ABC, abstractmethod

class BaseVisualizer(ABC):

def \_\_init\_\_(self, data):

self.data = data

@abstractmethod

def plot(self):

pass

def save\_plot(self, filename):

plt.savefig(filename)

print(f"Файл збережено як {filename}")

plt.close()

def show\_plot(self):

plt.show()

**Папка classes, файл data\_explorer.py:**

class DataExplorer:

def \_\_init\_\_(self, data):

self.data = data

def print\_extreme\_values(self):

print("Максимальні значення:\n", self.data.max())

print("Мінімальні значення:\n", self.data.min())

**Папка classes, файл data\_loader.py:**import pandas as pd

class DataLoader:

def \_\_init\_\_(self, filepath):

self.filepath = filepath

def load\_data(self):

data = pd.read\_csv(self.filepath)

return data

**Папка classes, файл histogram\_visualizer.py:**

import matplotlib.pyplot as plt

from classes.base\_visualizer import BaseVisualizer

class HistogramVisualizer(BaseVisualizer):

def plot(self):

if 'y\_column' not in self.data.columns:

print("Дані для візуалізації не знайдені!")

return

plt.figure()

plt.hist(self.data['y\_column'], bins=10, color='g', edgecolor='black')

plt.title("Гістограма")

plt.xlabel("Значення")

plt.ylabel("Частота")

self.save\_plot("histogram\_plot.png")

**Папка classes, файл error\_handler.py:**

class ErrorHandler:

def handle(self, error):

print(f"Помилка: {error}")

**Папка classes, файл line\_plot\_visualizer.py:**

import matplotlib.pyplot as plt

from classes.base\_visualizer import BaseVisualizer

class LinePlotVisualizer(BaseVisualizer):

def plot(self):

if 'x\_column' not in self.data.columns or 'y\_column' not in self.data.columns:

print("Дані для візуалізації не знайдені!")

return

plt.figure()

plt.plot(self.data['x\_column'], self.data['y\_column'], marker='o', color='b', linestyle='-')

plt.title("Лінійний графік")

plt.xlabel("X-вісь")

plt.ylabel("Y-вісь")

plt.grid(True)

self.save\_plot("line\_plot.png")

**Папка classes, файл subplots.py:**

import matplotlib.pyplot as plt

class Subplots:

def \_\_init\_\_(self, data):

self.data = data

def create\_subplots(self):

fig, axs = plt.subplots(1, 2, figsize=(10, 5))

axs[0].plot(self.data['x\_column'], self.data['y\_column'], marker='o', color='b', linestyle='-')

axs[1].hist(self.data['y\_column'])

plt.show()

**Файл main.py:**

from classes.data\_loader import DataLoader

from classes.data\_explorer import DataExplorer

from classes.line\_plot\_visualizer import LinePlotVisualizer

from classes.histogram\_visualizer import HistogramVisualizer

from classes.subplots import Subplots

def main():

# Завантаження даних

loader = DataLoader("data.csv")

data = loader.load\_data()

# Дослідження даних

explorer = DataExplorer(data)

explorer.print\_extreme\_values()

# Вибір візуалізації

visualizer = LinePlotVisualizer(data) # Використовуємо лінійний графік

visualizer.plot()

# Інша візуалізація

visualizer\_hist = HistogramVisualizer(data) # Використовуємо гістограму

visualizer\_hist.plot()

# Декілька піддіаграм

subplots = Subplots(data)

subplots.create\_subplots()

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

main()

**Файл data.csv:**

x\_column,y\_column

1,5

2,27

3,20

4,25

5,35

6,45

7,50

8,55

9,65

10,70

**ЛБ 9:**

**Файл runner.py:**

import os

import subprocess

from Calculator\_Pro\_Max\_beta\_v012.main import main as run\_lr1

from Calculator\_Pro\_Max\_beta\_v021.main import main as run\_lr2

from ASIIART\_Pro\_Max\_beta\_v033.main import main as run\_lr3

from ASIIART\_Pro\_Max\_beta\_v04.main import main as run\_lr4

from ASIIART\_Pro\_Max\_beta\_v051.main import main as run\_lr5

from Calculator\_Pro\_Max\_beta\_v06.main import main as run\_lr6

from API\_Pro\_Max\_beta\_v07.main import main as run\_lr7

from CSV\_Pro\_Max\_beta\_v08.main import main as run\_lr8

class LabRunner:

def \_\_init\_\_(self):

self.labs = {

1: run\_lr1,

2: run\_lr2,

3: run\_lr3,

4: run\_lr4,

5: run\_lr5,

6: run\_lr6,

7: run\_lr7,

8: run\_lr8,

}

def run\_lab(self, lab\_number):

try:

if lab\_number in self.labs:

print(f"Запускаємо лабораторну роботу №{lab\_number}...")

self.labs[lab\_number]()

else:

print("Некоректний номер лабораторної роботи.")

except AttributeError:

print(f"Помилка: лабораторна робота №{lab\_number} не має коректної функції main.")

def run\_tests(self, lab\_number):

"""

Запуск юніт-тестів для вказаної лабораторної роботи.

"""

test\_files = {

6: "Calculator\_Pro\_Max\_beta\_v06/test\_calculator.py",

7: "API\_Pro\_Max\_beta\_v07/tests.py",

}

test\_file = test\_files.get(lab\_number)

if not test\_file:

print(f"Для лабораторної роботи №{lab\_number} тести не знайдено.")

return

if os.path.exists(test\_file):

print(f"Запускаємо тести для лабораторної роботи №{lab\_number}...")

subprocess.run(["python", "-m", "unittest", test\_file])

else:

print(f"Файл з тестами не знайдено: {test\_file}.")

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

runner = LabRunner()

while True:

try:

print("\nМеню:")

print("1-8: Запуск лабораторної роботи")

print("9: Запуск тестів для 6-ї лабораторної")

print("10: Запуск тестів для 7-ї лабораторної")

print("0: Вихід")

choice = int(input("Введіть номер дії: "))

if choice == 0:

break

elif choice == 9:

runner.run\_tests(6)

elif choice == 10:

runner.run\_tests(7)

elif 1 <= choice <= 8:

runner.run\_lab(choice)

else:

print("Некоректний вибір.")

except ValueError:

print("Помилка: введіть коректне число.")

**Папка common:**

**Файл api\_tools.py:**

import requests

class APITools:

@staticmethod

def get(endpoint):

"""

Виконує GET-запит до API.

"""

try:

response = requests.get(endpoint)

response.raise\_for\_status()

return response.json()

except requests.RequestException as e:

print(f"Помилка запиту: {e}")

return None

**Файл data\_storage.py:**

import json

import csv

class DataStorage:

@staticmethod

def save\_to\_json(data, filepath):

"""

Зберігає дані у JSON файл.

"""

try:

with open(filepath, 'w', encoding='utf-8') as f:

json.dump(data, f, ensure\_ascii=False, indent=4)

print(f"Дані збережено у файл: {filepath}")

except Exception as e:

print(f"Помилка збереження у JSON: {e}")

@staticmethod

def load\_from\_json(filepath):

"""

Завантажує дані з JSON файлу.

"""

try:

with open(filepath, 'r', encoding='utf-8') as f:

return json.load(f)

except Exception as e:

print(f"Помилка завантаження з JSON: {e}")

return None

@staticmethod

def save\_to\_csv(data, filepath, fieldnames):

"""

Зберігає дані у CSV файл.

"""

try:

with open(filepath, 'w', encoding='utf-8', newline='') as f:

writer = csv.DictWriter(f, fieldnames=fieldnames)

writer.writeheader()

writer.writerows(data)

print(f"Дані збережено у CSV файл: {filepath}")

except Exception as e:

print(f"Помилка збереження у CSV: {e}")

@staticmethod

def load\_from\_csv(filepath):

"""

Завантажує дані з CSV файлу.

"""

try:

with open(filepath, 'r', encoding='utf-8') as f:

reader = csv.DictReader(f)

return list(reader)

except Exception as e:

print(f"Помилка завантаження з CSV: {e}")

return None

**Файл logger.py:**

import logging

class Logger:

def \_\_init\_\_(self, log\_file="application.log"):

logging.basicConfig(

filename=log\_file,

level=logging.INFO,

format="%(asctime)s - %(levelname)s - %(message)s"

)

def log\_info(self, message):

logging.info(message)

def log\_error(self, message):

logging.error(message)

**Файл validation.py:**

class Validation:

"""

Клас для перевірки введених даних.

"""

@staticmethod

def validate\_number(value):

"""

Перевірка, чи є значення числом.

"""

try:

return float(value)

except ValueError:

print("Помилка: введене значення не є числом.")

return None

@staticmethod

def validate\_non\_empty\_string(value):

"""

Перевірка, чи є значення непорожнім рядком.

"""

if isinstance(value, str) and value.strip():

return value

print("Помилка: рядок порожній або некоректний.")

return None

**Результат виконання:**

****

**Висновок:** на цій лабораторній роботі я об’єднав і доопрацював всі лабораторні роботи в один великий проект, який працює через runner.py