МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж



Лабораторна робота №7

з дисципліни

Спеціалізовані мови програмування

на тему

Робота з API та веб-сервісами

Виконала:

ст. гр. ІТ-32

Ольга ЧИГИРИК

Прийняв

доцент каф. ІСМ:

Сергій ЩЕРБАК

|  |  |
| --- | --- |
| **Балів** | **Дата** |
|  |  |

Львів-2023

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

**Хід роботи:**

**Завдання 1: Вибір провайдера API**

Виберіть надійний API, який надає через HTTP необхідні дані для віддаленого зберігання, вивантаження або реалізуйте свій. Для прикладу це може бути jsonplaceholder.org

**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.

**Main.py**

from api\_client7 import APIClient  
from console7 import ConsOleUI  
from Scripts.all\_python\_labs.shared\_lib.calculation\_functions import CalculationFunctions  
from Scripts.all\_python\_labs.shared\_lib.memory\_manager import MemoryManager  
from Scripts.all\_python\_labs.shared\_lib.history\_logger import HistoryLogger  
from Scripts.all\_python\_labs.shared\_lib.data\_manager import DataManager  
from Scripts.all\_python\_labs.shared\_lib.input\_validator import InputValidator  
# from history\_logging7 import HistoryLogger  
# from input\_parser7 import InputValidator  
  
  
def main():  
 api\_client = APIClient()  
 console\_ui = ConsOleUI(api\_client)  
 input\_validator = InputValidator()  
 history\_logger = HistoryLogger()  
  
 while True:  
 choice = console\_ui.display\_menu()  
  
 if choice == '1':  
 date\_input = ''  
 while not input\_validator.validate\_date(date\_input):  
 date\_input = input("Enter date (DD-MM-YYYY): ")  
 if not input\_validator.validate\_date(date\_input):  
 print("Invalid date format. Try again.")  
  
 phone\_input = ''  
 while not input\_validator.validate\_phone\_number(phone\_input):  
 phone\_input = input("Enter phone number: ")  
 if not input\_validator.validate\_phone\_number(phone\_input):  
 print("Invalid phone number format. Try again.")  
  
 display\_format = ''  
 while not input\_validator.validate\_display\_format(display\_format):  
 display\_format = console\_ui.choose\_display\_format()  
 if not input\_validator.validate\_display\_format(display\_format):  
 print("Invalid display format. Try again.")  
  
 num\_posts\_str = ''  
 num\_posts = 0  
 while not num\_posts:  
 num\_posts\_str = console\_ui.choose\_number\_of\_posts()  
 num\_posts = input\_validator.validate\_number(num\_posts\_str)  
 if not num\_posts:  
 print("Invalid number of posts. Enter a positive integer.")  
  
 posts = api\_client.get\_posts()  
 color\_choice = console\_ui.choose\_color()  
 color\_code = console\_ui.get\_color\_code(color\_choice)  
  
 if display\_format == '2': # Table format  
 style\_choice = console\_ui.choose\_table\_style()  
 console\_ui.display\_posts(posts, display\_format, num\_posts, color\_code, style\_choice)  
 else: # List format  
 list\_style\_choice = console\_ui.choose\_list\_style()  
 console\_ui.display\_list(posts, num\_posts, color\_code, list\_style\_choice)  
  
 ask\_to\_save = console\_ui.ask\_to\_save\_data(posts)  
 if ask\_to\_save:  
 history\_logger.log\_history("Saved posts", "Success")  
  
 elif choice == '2':  
 print("Exiting the program...")  
 break  
  
 elif choice == '3':  
 print("Request History:")  
 print(history\_logger.read\_history())  
  
 else:  
 print("Unknown command, try again.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Console.py**

# from prettytable import PrettyTable  
# from data\_management7 import DataManager  
# from input\_parser7 import InputValidator  
from Scripts.all\_python\_labs.shared\_lib.calculation\_functions import CalculationFunctions  
from Scripts.all\_python\_labs.shared\_lib.memory\_manager import MemoryManager  
from Scripts.all\_python\_labs.shared\_lib.history\_logger import HistoryLogger  
from Scripts.all\_python\_labs.shared\_lib.api\_client import APIClient  
from Scripts.all\_python\_labs.shared\_lib.data\_manager import DataManager  
from Scripts.all\_python\_labs.shared\_lib.input\_validator import InputValidator  
from prettytable import PrettyTable  
  
  
class ConsOleUI:  
 def \_\_init\_\_(self, api\_client):  
 self.api\_client = api\_client  
  
 def display\_menu(self):  
 print("1: Retrieve Posts")  
 print("2: Exit")  
 print("3: View History")  
 return input("Your choice: ")  
  
 def choose\_display\_format(self):  
 print("Select display format:")  
 print("1: List")  
 print("2: Table")  
 return input("Your choice (1/2): ")  
  
 def choose\_number\_of\_posts(self):  
 return input("Enter the number of posts to display: ")  
  
 def choose\_table\_style(self):  
 print("Select table style:")  
 print("1: Plain")  
 print("2: Line-separated")  
 return input("Your choice (1/2): ")  
  
 def choose\_list\_style(self):  
 print("Select list style:")  
 print("1: Plain")  
 print("2: Numbered")  
 return input("Your choice (1/2): ")  
  
 def ask\_to\_save\_data(self, data):  
 user\_choice = input("Do you want to save the data to a file? (yes/no): ").lower()  
 if user\_choice in ['yes', 'так']:  
 filename = input("Enter file name (default: data.txt): ")  
 filename = filename if filename else 'data.txt'  
 DataManager.save\_data\_to\_file(data, filename)  
 print(f"Data saved to file: {filename}")  
 else:  
 print("Save cancelled.")  
  
 def choose\_color(self):  
 print("Choose a color:")  
 print("1: Gray")  
 print("2: Red")  
 print("3: Green")  
 print("4: Yellow")  
 print("5: Blue")  
 print("6: Pink")  
 print("7: Cyan")  
 print("8: White")  
 return input("Your choice (1-8): ")  
  
 def get\_color\_code(self, color\_choice):  
 colors = {  
 '1': '\033[30m', # Gray  
 '2': '\033[31m', # Red  
 '3': '\033[32m', # Green  
 '4': '\033[33m', # Yellow  
 '5': '\033[34m', # Blue  
 '6': '\033[35m', # Pink  
 '7': '\033[36m', # Cyan  
 '8': '\033[37m', # White  
 }  
 return colors.get(color\_choice, '\033[0m') # Default color  
  
 def display\_list(self, posts, num\_posts, color\_code, list\_style\_choice):  
 num\_posts = int(num\_posts)  
 posts = posts[:num\_posts]  
  
 for index, post in enumerate(posts, start=1):  
 if list\_style\_choice == '2': # Numbered list  
 print(f"{color\_code}{index}. ID: {post['id']}, Title: {post['title']}\033[0m")  
 else: # Plain list  
 print(f"{color\_code}ID: {post['id']}, Title: {post['title']}\033[0m")  
  
 def display\_posts(self, posts, display\_format, num\_posts, color\_code, table\_style\_choice):  
 num\_posts = int(num\_posts)  
 posts = posts[:num\_posts]  
  
 if display\_format == '2': # Table format  
 table = PrettyTable()  
 table.field\_names = [f"{color\_code}ID\033[0m", f"{color\_code}Title\033[0m"]  
  
 for post in posts:  
 table.add\_row([post['id'], post['title']])  
  
 if table\_style\_choice == '2': # Style with lines  
 table.border = True  
 table.header = True  
 table.horizontal\_char = '-'  
 table.vertical\_char = '|'  
 table.junction\_char = '+'  
 else: # Default style  
 table.border = False  
 table.header = True  
 table.horizontal\_char = ' '  
 table.vertical\_char = ' '  
 table.junction\_char = ' '  
  
 print(table)  
 else: # List format  
 self.display\_list(posts, num\_posts, color\_code, table\_style\_choice)

**Data\_manager.py**

import json  
  
class DataManager:  
 @staticmethod  
 def save\_data\_to\_file(data, filename='data.txt'):  
 *"""Saves data to a specified file in JSON format."""* with open(filename, 'w', encoding='utf-8') as file:  
 json.dump(data, file, ensure\_ascii=False, indent=4)  
 print(f"Data successfully saved to {filename}.")  
  
 @staticmethod  
 def load\_data\_from\_file(filename='data.txt'):  
 *"""Loads data from a specified file."""* try:  
 with open(filename, 'r', encoding='utf-8') as file:  
 return json.load(file)  
 except FileNotFoundError:  
 print(f"File {filename} not found.")  
 return None  
 except json.JSONDecodeError:  
 print(f"Error decoding JSON from {filename}.")  
 return None

**History\_logging.py**

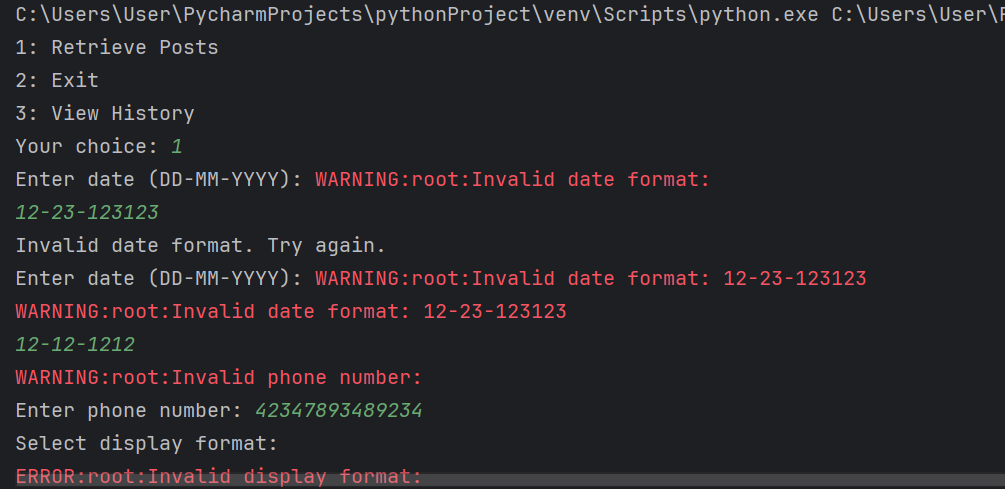
import datetime  
  
class HistoryLogger:  
 LOG\_FILE = 'history\_log7.txt'  
  
 @staticmethod  
 def log\_history(user\_input, result):  
 *"""Log user input and result to a file."""* with open(HistoryLogger.LOG\_FILE, 'a', encoding='utf-8') as file:  
 timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
 log\_entry = f"{timestamp} - Input: {user\_input}, Result: {result}\n"  
 file.write(log\_entry)  
  
 @staticmethod  
 def read\_history():  
 *"""Read and return the history from the log file."""* try:  
 with open(HistoryLogger.LOG\_FILE, 'r', encoding='utf-8') as file:  
 return file.read()  
 except FileNotFoundError:  
 return "History not found."

**Input\_parser.py**

import re  
from datetime import datetime  
  
class InputValidator:  
 @staticmethod  
 def validate\_number(input\_str):  
 *"""Checks if the input string is a positive integer."""* try:  
 number = int(input\_str)  
 if number > 0:  
 return number  
 else:  
 raise ValueError("The number must be greater than 0.")  
 except ValueError:  
 return None  
  
 @staticmethod  
 def validate\_display\_format(input\_str):  
 *"""Checks if the input display format is valid (either '1' or '2')."""* if input\_str in ['1', '2']:  
 return input\_str  
 else:  
 return None  
  
 @staticmethod  
 def validate\_date(input\_str):  
 *"""Checks if the input is a valid date in the format DD-MM-YYYY."""* try:  
 datetime.strptime(input\_str, '%d-%m-%Y')  
 return input\_str  
 except ValueError:  
 return None  
  
 @staticmethod  
 def validate\_phone\_number(input\_str):  
 *"""Checks if the input is a valid phone number."""* pattern = r'\+?\d{1,3}?[-.\s]?\(?\d{2,3}\)?[-.\s]?\d{3}[-.\s]?\d{4}'  
 if re.match(pattern, input\_str):  
 return input\_str  
 else:  
 return None

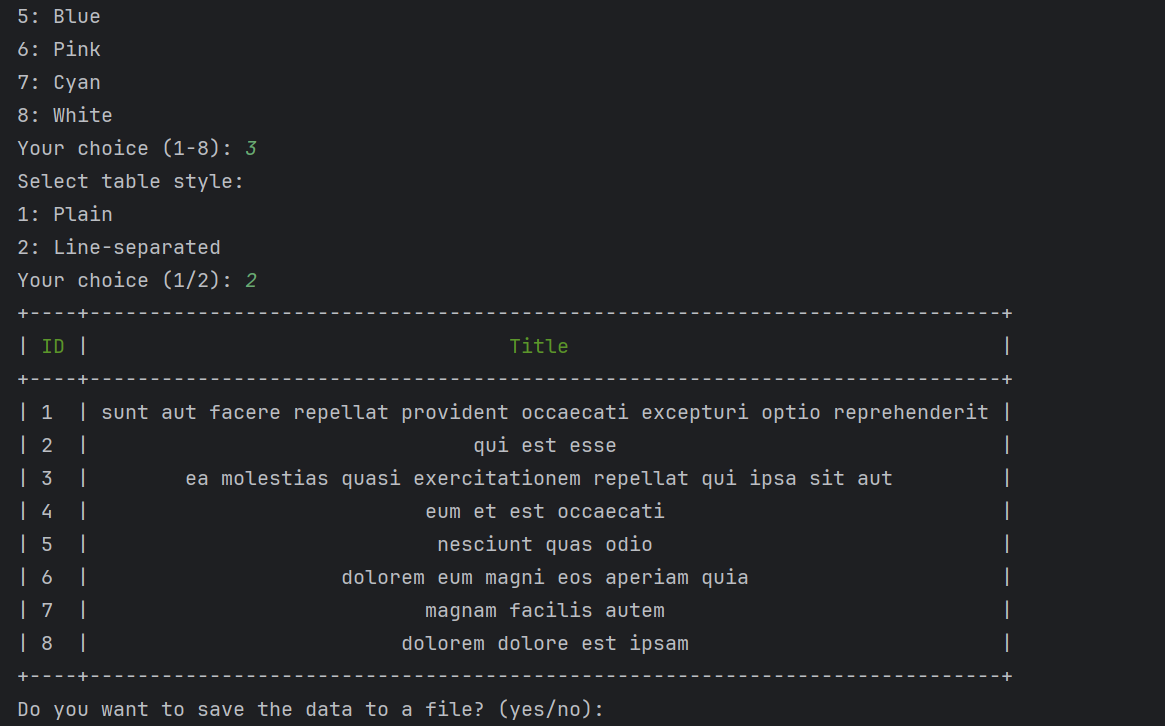
**Api\_client.py**

import requests  
  
class APIClient:  
 BASE\_URL = "https://jsonplaceholder.typicode.com/posts"  
  
 @staticmethod  
 def get\_posts():  
 *"""Retrieve posts from the API."""* try:  
 response = requests.get(APIClient.BASE\_URL)  
 response.raise\_for\_status() # Raises an error for response codes 4xx/5xx  
 return response.json()  
 except requests.RequestException as e:  
 return f"Error during request: {e}"



***Рис. 1 Результат виконання завдання*

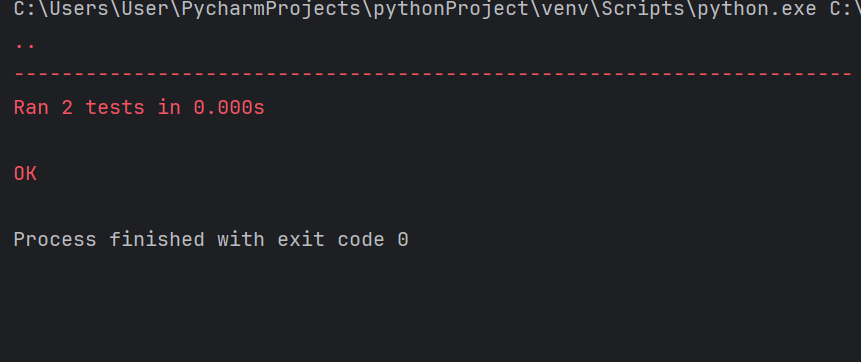
*Рис. 2 Результат виконання завдання*

**

*Рис. 3 Результат виконання завдання*

**Test\_console.py**

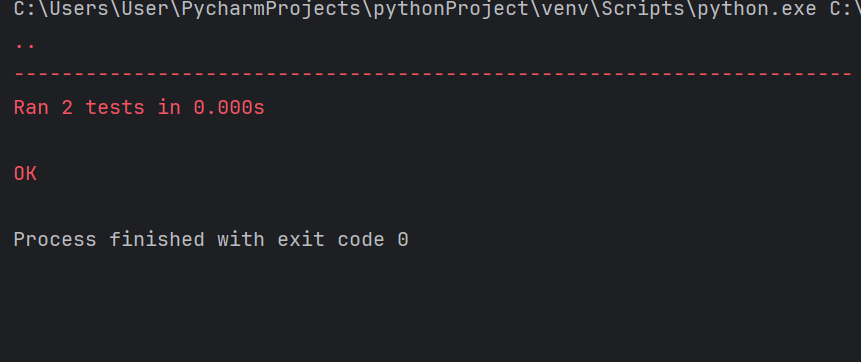
import unittest  
from Scripts.all\_python\_labs.lab\_python\_7.console7 import get\_color\_code  
  
class TestConsole7(unittest.TestCase):  
  
 def test\_get\_color\_code\_valid(self):  
 self.assertEqual(get\_color\_code("1"), '\033[30m') # Testing for grey color code  
 self.assertEqual(get\_color\_code("8"), '\033[37m') # Testing for white color code  
  
 def test\_get\_color\_code\_invalid(self):  
 self.assertEqual(get\_color\_code("9"), '\033[0m') # Testing for default color code  
 self.assertEqual(get\_color\_code("abc"), '\033[0m') # Testing for non-numeric input  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()  
import unittest  
  
  
class MyTestCase(unittest.TestCase):  
 def test\_something(self):  
 self.assertEqual(True, False) # add assertion here  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()



*Рис. 4 Результат виконання юніттесту*

**Test\_imput\_perser.py**

import unittest  
from Scripts.all\_python\_labs.lab\_python\_7.input\_parser7 import validate\_number  
  
  
class TestInputParser7(unittest.TestCase):  
  
 def test\_validate\_number\_valid(self):  
 self.assertEqual(validate\_number("10"), 10)  
 self.assertEqual(validate\_number("1"), 1)  
  
 def test\_validate\_number\_invalid(self):  
 self.assertIsNone(validate\_number("0"))  
 self.assertIsNone(validate\_number("-5"))  
 self.assertIsNone(validate\_number("abc"))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**

*Рис. 5 Результат виконання юніттесту*

**Висновок:** Виконавши ці завдання, я створила проект, який надав цінний досвід роботи з API, дизайном користувацького інтерфейсу, валідації введення, обробки помилок та тестування.