МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №7

з курсу

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

Виконав студент

групи ІТ-21сп

**Гузар А.І.**

Прийняв

**Щербак С.С.**

Львів - 2023

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

**План 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: Відображення результатів**

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.

Клас ConsoleInterface:

import sys

from DogAPI import DogAPI

from DisplayDogAPI import DisplayDogApi

import signal

from colorama import Fore

from DogDisplay import DogDisplay

class ConsoleInterface:

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

# Initialize the ConsoleInterface with an empty history list

self.history = []

def signal\_handler(sig, frame):

# Signal handler for Ctrl+C interrupt

print("You pressed Ctrl+C!")

sys.exit(0)

def choose\_display\_format():

# Prompt the user to choose a display format (Table or List)

while True:

print("Choose a display format:")

print("1. Table")

print("2. List")

format\_choice = input("Enter the number of the display format: ")

if format\_choice in ("1", "2"):

return format\_choice

else:

print("Invalid display format. Please enter 1 or 2.")

def choose\_color():

# Prompt the user to choose a color (Red, Green, or Yellow)

while True:

print("Choose a color:")

print("1. Red")

print("2. Green")

print("3. Yellow")

color\_choice = input("Enter the number of the color: ")

if color\_choice == "1":

return Fore.RED

elif color\_choice == "2":

return Fore.GREEN

elif color\_choice == "3":

return Fore.YELLOW

else:

print("Invalid color choice. Please enter 1, 2, or 3.")

def start\_menu():

# Display the start menu options and prompt the user to choose an option

print("Hello! It's dog API. Choose the option:")

print("1. Continue")

print("2. Exit")

option = input("Enter the number of the option: ")

return option

def main():

# Main function to interact with the user and perform operations

api\_key = "live\_Qm70bYO91yDzUWHnjK8SpliWvFnkRCPl5lS6f8THqR0qDkzP84oTm4Y8RF0WFJvm"

dog\_api = DogAPI(api\_key)

interface = ConsoleInterface()

while True:

user\_option = start\_menu()

if user\_option == "1":

breeds = DisplayDogApi.get\_all\_breeds()

color = choose\_color()

format\_choice = choose\_display\_format()

if format\_choice == "1":

DogDisplay.display\_table(breeds, color)

elif format\_choice == "2":

DogDisplay.display\_list(breeds, color)

save\_choice = input("Do you want to save the data? (y/n): ")

if save\_choice.lower() == 'y':

save\_format = input("Choose a save format:\n1. JSON\n2. CSV\n3. TXT\nEnter the number of the save format: ")

save\_filename = input("Enter the filename: ")

if save\_format == "1":

DogDisplay.save\_to\_json(breeds, save\_filename)

elif save\_format == "2":

DogDisplay.save\_to\_csv(breeds, save\_filename)

elif save\_format == "3":

DogDisplay.save\_to\_txt(breeds, save\_filename)

else:

print("Invalid save format. Please enter 1, 2, or 3.")

repeat\_choice = input("Do you want to perform another operation? (y/n): ")

if repeat\_choice.lower() != 'y':

break

elif user\_option == "2":

print("Exiting the program. Goodbye!")

break

else:

print("Invalid option. Please enter 1 or 2.")

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

# Set up a signal handler for Ctrl+C

signal.signal(signal.SIGINT, signal\_handler)

# Call the main function

main()

Клас DogApi:

from colorama import init  
  
# Initialize colorama with autoreset to automatically reset colors after each print statement  
init(autoreset=True)  
  
class DogAPI:  
 # Base URL for The Dog API  
 base\_url = "https://api.thedogapi.com/v1"  
  
 def \_\_init\_\_(self, api\_key):  
 # Initialize DogAPI object with the provided API key  
 self.headers = {"x-api-key": api\_key}

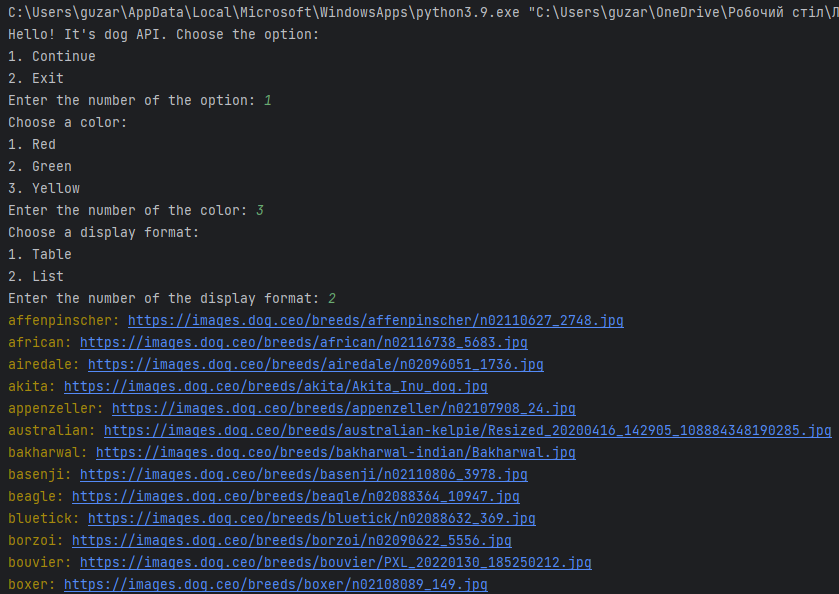
Клас DogDisplay:

import csv  
import json  
from io import BytesIO  
  
import requests  
from PIL import Image  
from tabulate import tabulate  
from colorama import Fore  
  
from Lab7.DisplayDogAPI import DisplayDogApi  
  
  
class DogDisplay:  
 @staticmethod  
 def display\_table(data, color):  
 # Display data in a table format with headers and colors  
 headers = [Fore.RESET + color + "DogBreed", Fore.RESET + color + "PictureUrl"]  
 rows = [(color + breed, DisplayDogApi.get\_random\_image(breed)) for breed in data]  
 table = tabulate(rows, headers, tablefmt="grid")  
 print(table)  
  
 @staticmethod  
 def display\_list(data, color):  
 # Display data as a list with colors  
 for breed in data:  
 print(color + f"{breed}: {DisplayDogApi.get\_random\_image(breed)}")  
  
 @staticmethod  
 def display\_image(image\_url):  
 # Display an image given its URL  
 response = requests.get(image\_url)  
 img = Image.open(BytesIO(response.content))  
 img.show()  
  
 @staticmethod  
 def remove\_color\_codes(text):  
 # Remove color codes from the given text  
 while '\033[' in text:  
 start = text.find('\033[')  
 end = text.find('m', start)  
 if end != -1:  
 text = text[:start] + text[end + 1:]  
 else:  
 break  
 return text  
  
 @staticmethod  
 def save\_to\_json(data, filename):  
 # Save data to a JSON file  
 data\_to\_save = [{"DogBreed": breed, "PictureUrl": DisplayDogApi.get\_random\_image(breed)} for breed in data]  
 with open(filename, 'w') as file:  
 json.dump(data\_to\_save, file, indent=2)  
 print(f"Data saved to {filename} in JSON format.")  
  
 @staticmethod  
 def save\_to\_csv(data, filename):  
 # Save data to a CSV file  
 with open(filename, 'w', newline='') as file:  
 writer = csv.writer(file)  
 writer.writerow([Fore.RESET + "DogBreed", Fore.RESET + "PictureUrl"])  
 for breed in data:  
 writer.writerow([DogDisplay.remove\_color\_codes(breed), DogDisplay.remove\_color\_codes(DisplayDogApi.get\_random\_image(breed))])  
 print(f"Data saved to {filename} in CSV format.")  
  
 @staticmethod  
 def save\_to\_txt(data, filename):  
 # Save data to a TXT file  
 with open(filename, 'w') as file:  
 for breed in data:  
 file.write(f"{DogDisplay.remove\_color\_codes(breed)}: {DogDisplay.remove\_color\_codes(DisplayDogApi.get\_random\_image(breed))}\n")  
 print(f"Data saved to {filename} in TXT format.")

Клас Test:

import unittest  
from unittest.mock import patch, MagicMock  
from io import StringIO  
  
from Lab7.ConsoleInterface import choose\_display\_format  
from Lab7.DisplayDogAPI import DisplayDogApi  
  
class TestDisplayDogApi(unittest.TestCase):  
 @patch('requests.get')  
 def test\_get\_all\_breeds(self, mock\_requests\_get):  
 # Create a mock response with sample data  
 mock\_response = MagicMock()  
 mock\_response.json.return\_value = {"message": {"breed1": {}, "breed2": {}}}  
 # Set the mock response for the requests.get method  
 mock\_requests\_get.return\_value = mock\_response  
  
 # Call the method under test  
 breeds = DisplayDogApi.get\_all\_breeds()  
  
 # Assert that the method returns the expected list of breeds  
 self.assertEqual(list(breeds), ["breed1", "breed2"])  
  
class TestConsoleInterface(unittest.TestCase):  
 @patch('builtins.input', side\_effect=['3'])  
 def test\_invalid\_display\_format(self, mock\_input):  
 # Test that an invalid display format input raises a ValueError or StopIteration  
 with self.assertRaises((ValueError, StopIteration)):  
 choose\_display\_format()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Run the test cases if the script is executed as the main module  
 unittest.main()

На рисунку 1 зображено результат виконання програми.



*Рис1. Результат запиту*



*Рис.2 Результат збереження.*

**Висновок:** під час виконання лабораторної роботи я навчився cтворювати консольного об’єктно - орієнтованого додатка з використанням API