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

Кузьо О.Р.

Прийняв

Щербак С.С.

Львів – 2023

Лабораторна робота №7

**Мета:** створення консольного обʼєктно-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.

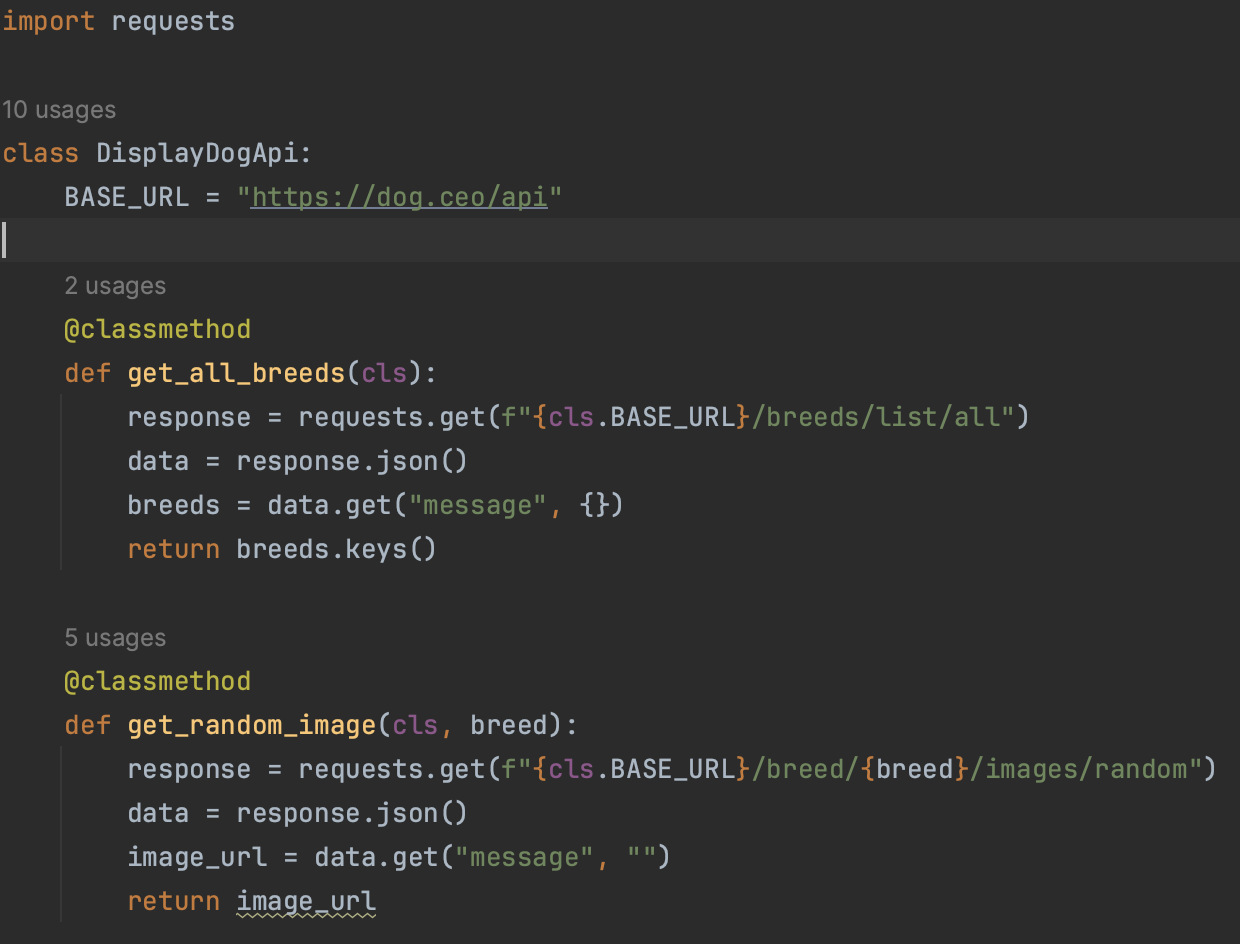


Рис.1. інтеграція API

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.

@staticmethod

def choose\_display\_format():

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.")

@staticmethod

def choose\_color():

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.")

@staticmethod

def start\_menu():

print("Hello! Choose the option:")

print("1. Continue")

print("2. Exit")

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

return option

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

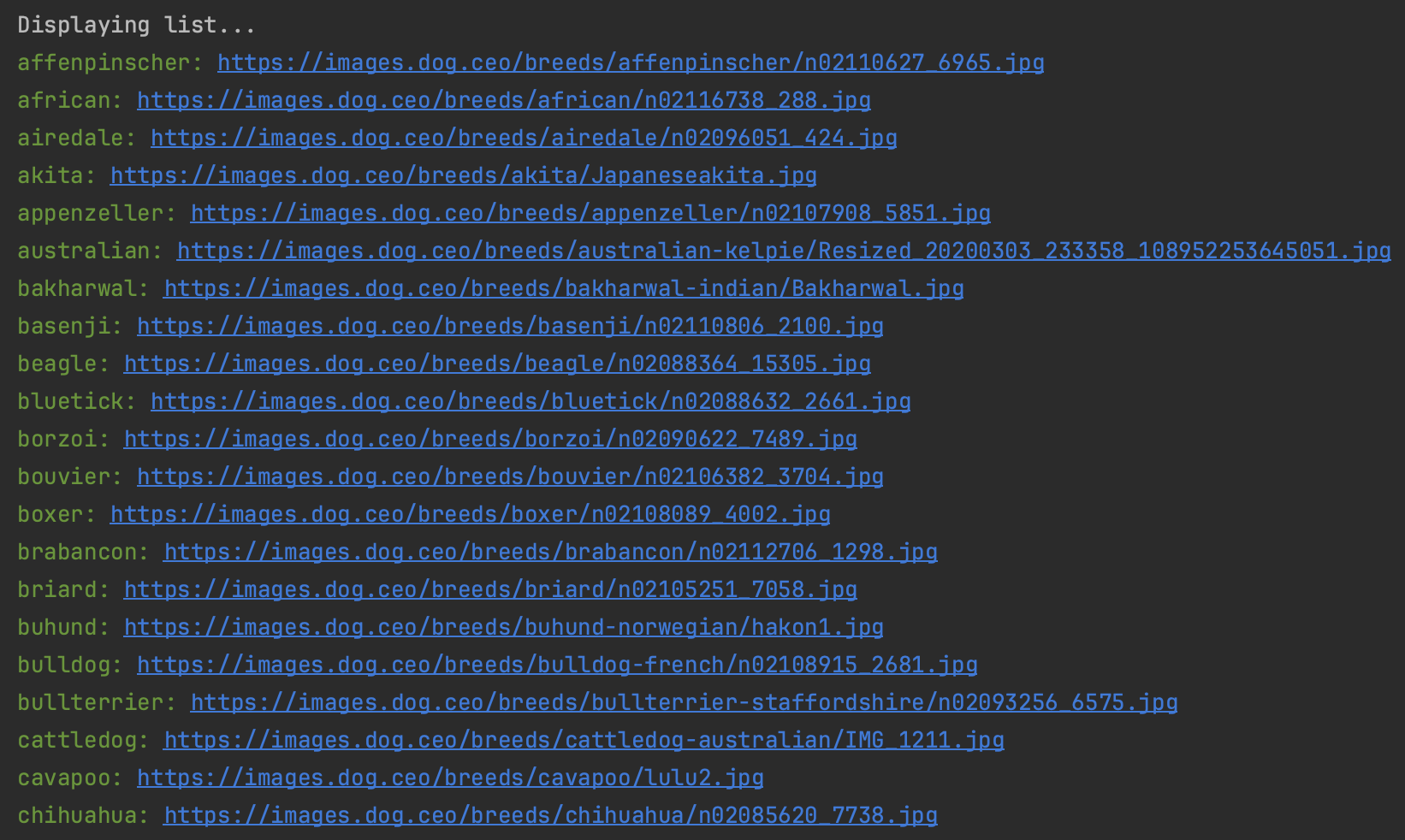


Рис.2. відображення результатів

Завдання 6: Збереження даних

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

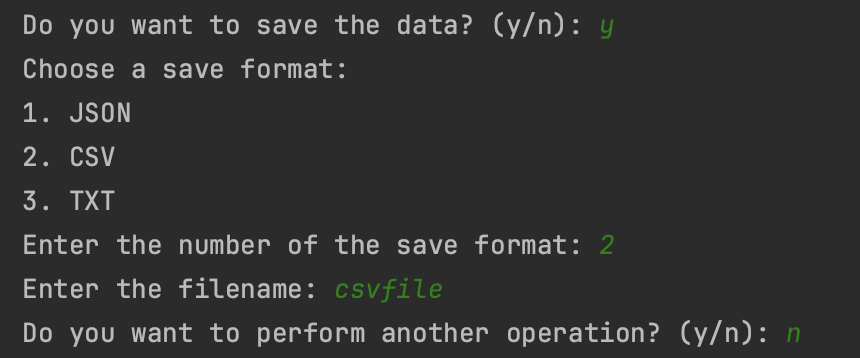


Рис.3. збереження даних

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.

import unittest

from unittest.mock import patch, MagicMock

from labs.classes.Lab7.display\_dog\_api import DisplayDogApi

from labs.UI.Menu\_Builder.Lab7.lab\_menu7 import MenuLab7

class TestDisplayDogApi(unittest.TestCase):

"""

A test class for testing the DisplayDogApi and MenuLab7 classes.

Methods:

- test\_get\_all\_breeds: Test the get\_all\_breeds method of DisplayDogApi class.

- test\_invalid\_display\_format: Test the choose\_display\_format method of MenuLab7 class.

"""

@patch('requests.get')

def test\_get\_all\_breeds(self, mock\_requests\_get):

"""

Test the get\_all\_breeds method of DisplayDogApi class.

This method uses the @patch decorator to mock the requests.get method

and simulate the API response.

The mock\_response is configured to return a dictionary simulating the API response.

The test then calls DisplayDogApi.get\_all\_breeds() and checks if the returned breeds

match the expected list.

"""

mock\_response = MagicMock()

mock\_response.json.return\_value = {"message": {"breed1": {}, "breed2": {}}}

mock\_requests\_get.return\_value = mock\_response

breeds = DisplayDogApi.get\_all\_breeds()

self.assertEqual(list(breeds), ["breed1", "breed2"])

@patch('builtins.input', side\_effect=['3'])

def test\_invalid\_display\_format(self):

"""

Test the choose\_display\_format method of MenuLab7 class.

This method uses the @patch decorator to mock the builtins.input method

and simulate user input ('3').

The test then calls MenuLab7.choose\_display\_format() and checks if the expected

exception (ValueError or StopIteration) is raised.

"""

with self.assertRaises((ValueError, StopIteration)):

MenuLab7.choose\_display\_format()

**Висновок:** під час опрацювання лабораторної роботи я створив консольний обʼєктно-орієнтований додаток з використанням API.