Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



**Звіт**

до лабораторної роботи №8

З дисципліни: «Кросплатформні засоби програмування»

На тему: «ФАЙЛИ ТА ВИКЛЮЧЕННЯ У PYTHON»

**Варіант 3**

Виконав: ст. гр. КІ-305

Гнідець В.М.

Прийняв:

Іванов Ю.С.

Львів 2023

**Мета роботи:** оволодіти навиками використання засобів мови Python для роботи з файлами.

**Завдання:**

1.Написати та налагодити програму на мові Python згідно варіанту. Програма має задовольняти наступним вимогам:

• програма має розміщуватися в окремому модулі;

• програма має реалізувати функції читання/запису файлів у текстовому і двійковому форматах результатами обчислення виразів згідно варіанту;

• програма має містити коментарі.

2. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.

3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.

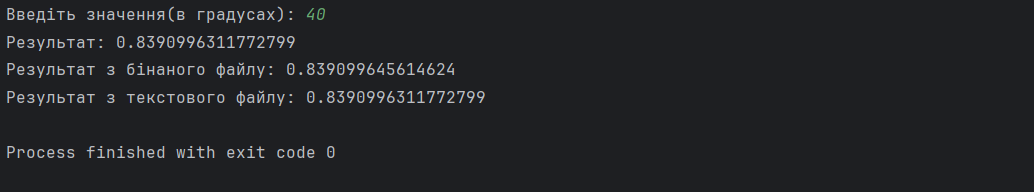
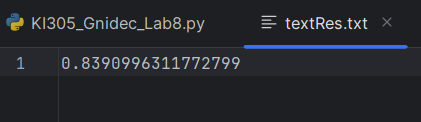
4. Дати відповідь на контрольні запитання.

**Варіант 3:** y=sin(x)/cos(x)

**Код програми**

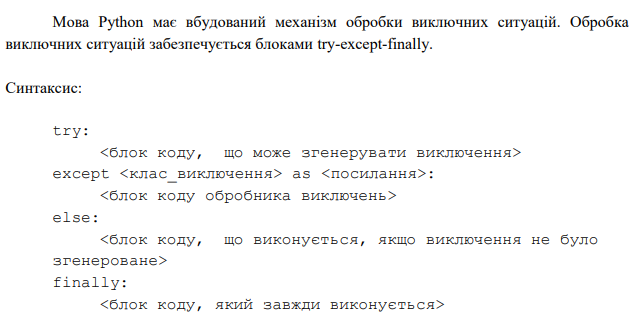
import os  
import struct  
import math  
  
# Функція для запису результату у текстовому форматі  
def writeResTxt(fName, result):  
 with open(fName, 'w') as f:  
 f.write(str(result))  
  
# Функція для читання результату з текстового файлу  
def readResTxt(fName):  
 result = 0.0  
 try:  
 if os.path.exists(fName):  
 with open(fName, 'r') as f:  
 result = f.read()  
 else:  
 raise FileNotFoundError(f"Файл {fName} не знайдено.")  
 except FileNotFoundError as e:  
 print(e)  
 return result  
  
# Функція для запису результату у двійковому форматі  
def writeResBin(fName, result):  
 with open(fName, 'wb') as f:  
 # Використовуємо struct.pack для збереження числа у двійковому форматі  
 f.write(struct.pack('f', result))  
  
# Функція для читання результату з двійкового файлу  
def readResBin(fName):  
 result = 0.0  
 try:  
 if os.path.exists(fName):  
 with open(fName, 'rb') as f:  
 # Використовуємо struct.unpack для розпакування числа з двійкового формату  
 result = struct.unpack('f', f.read())[0]  
 else:  
 raise FileNotFoundError(f"Файл {fName} не знайдено.")  
 except FileNotFoundError as e:  
 print(e)  
 return result  
  
# Функція для обчислення виразу sin(x)/cos(x) в радіанах  
EPSILON = 1e-10 # Дуже мале число для порівняння з нулем  
  
def calculate(degrees):  
 radians = math.radians(degrees) # Конвертуємо градуси в радіани  
  
 if abs(math.cos(radians)) < EPSILON:  
 # Викидаємо виключення, якщо cos(x) дуже близький до нуля  
 raise ValueError("Косинус дуже близький до 0, виникає ділення на 0.")  
  
 return math.sin(radians) / math.cos(radians)  
  
# Головна частина програми  
if \_\_name\_\_ == "\_\_main\_\_":  
 try:  
 # Зчитуємо вхідні дані від користувача (градуси)  
 data = float(input("Введіть значення(в градусах): "))  
 # Обчислюємо результат у радіанах  
 result = calculate(data)  
 # Виводимо результат у радіанах  
 print(f"Результат: {result}")  
 # Записуємо результат у файли  
 writeResTxt("textRes.txt", result)  
 writeResBin("binRes.bin", result)  
 # Читаємо результат із двійкового та текстового файлів та виводимо їх  
 print("Результат з бінаного файлу: {0}".format(readResBin("binRes.bin")))  
 print("Результат з текстового файлу: {0}".format(readResTxt("textRes.txt")))  
 except ValueError as e:  
 # Виводимо повідомлення про помилку, якщо виникає ValueError  
 print(e)  
 except FileNotFoundError as e:  
 # Виводимо повідомлення про помилку, якщо файл не знайдено  
 print(e)  
 except Exception as e:  
 # Виводимо загальне повідомлення про будь-яку іншу помилку  
 print("Виникла помилка: {0}".format(e))

**Результат роботи програми:**

**** ****

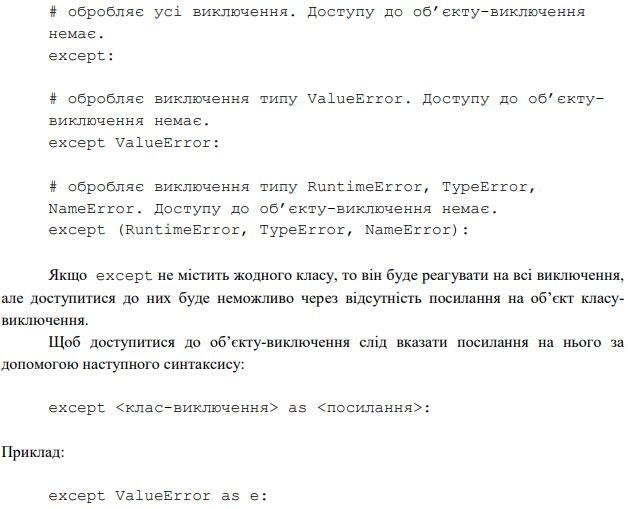
**Контрольні запитання**

***1. За допомогою якої конструкції у мові Python обробляються виключні ситуації?***

******

***2. Особливості роботи блоку except?***

Блок except не є обов’язковим, за умови, що визначено блок finally. Він містить код обробки виключної ситуації. Він може приймати перелік класів-виключень при генерації об’єктів яких буде виконане тіло даного блоку або бути порожнім:

******

У даному прикладі до об’єкту-виключення можна дістатися за допомогою посилання e. Об’єкт-виключення прийнято називати e, ex, exc або err.

Блоків except може не бути, бути один або бути багато. Якщо блоків багато, то кожен блок роблять таким, щоб він обробляв певний тип чи типи виключень. Блок except без параметрів, якщо використовується разом з іншими блоками except з параметрами, прийнято ставити в кінці послідовності блоків except, щоб він обробляв усі виключення які не були оброблені попередніми блоками except з параметрами.

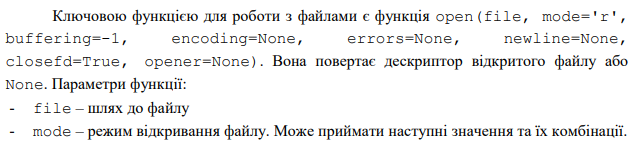
При необхідності блок except може мати необов’язковий блок else, який виконується, якщо виключення даного типу не було згенероване і блок except не виконувався.

Блок finally виконується завжди, якщо він є присутній. Цей блок може бути відсутнім, якщо присутній блок except.

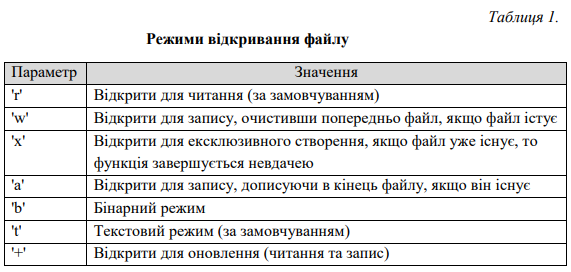
***3. Яка функція використовується для відкривання файлів у Python?***

У мові програмування Python для відкриття файлів використовується функція **open()**. Функція **open()** дозволяє вам вказати ім'я файлу, його режим відкриття та інші параметри, які дозволяють вам зчитувати, записувати або взаємодіяти з файлами в залежності від ваших потреб.

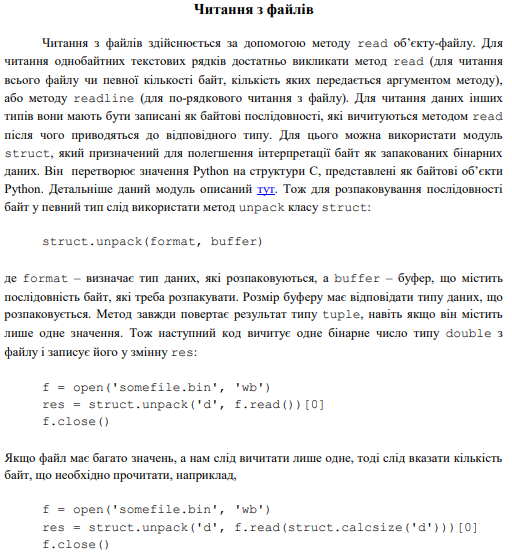
***4. Особливості використання функції open?***

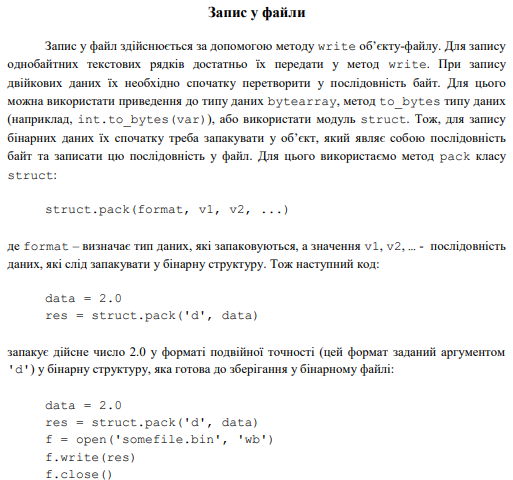
******

***5. В яких режимах можна відкрити файл?***

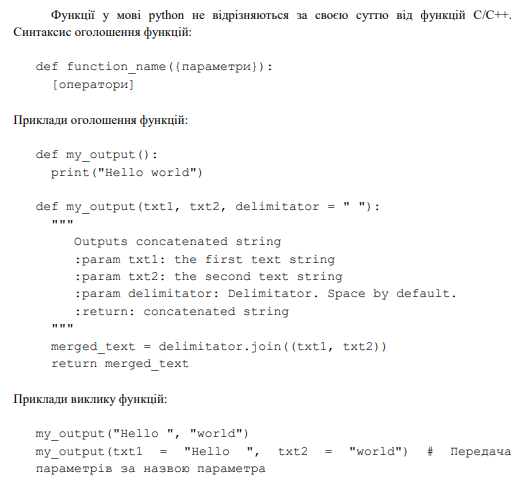
******

***6. Як здійснити читання і запис файлу?***

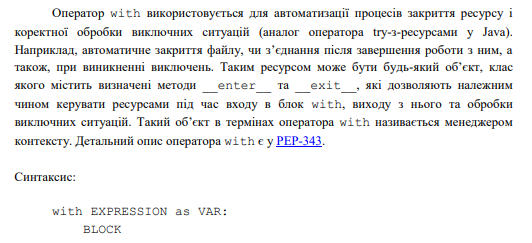
******

******

***7. Особливості функцій у мові Python?***

******

***8. Для чого призначенйи оператор with?***

******

***9. Які вимоги ставляться до об’єктів, що передаються під контроль оператору with?***

У мові програмування Python об'єкти, які передаються під контроль оператору **with**, повинні відповідати двом основним вимогам:

1. Об'єкт має підтримувати методи **\_\_enter\_\_()** і **\_\_exit\_\_()**:
   * Метод **\_\_enter\_\_()** викликається, коли контекстний блок починає виконуватися (**with** блок).
   * Метод **\_\_exit\_\_()** викликається при завершенні виконання контекстного блоку, навіть якщо виникла помилка.
2. Метод **\_\_exit\_\_()** має приймати три аргументи: **self**, **exc\_type**, і **exc\_value**. Де:
   * **self**: Посилання на об'єкт, який передається під контроль **with**.
   * **exc\_type**: Тип винятка (помилки), якщо така виникла. Якщо виняток не виник, цей аргумент буде **None**.
   * **exc\_value**: Об'єкт винятка (помилки), якщо така виникла. Якщо виняток не виник, цей аргумент буде **None**.

Приклад простого контекстного менеджера, який відповідає цим вимогам, може виглядати так:

class MyContextManager:

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

print("Entering the context")

# Можна виконувати ініціалізацію або відкривати ресурси

return self # Повертаємо об'єкт для використання в блоку 'with'

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

print("Exiting the context")

# Можна виконувати завершальні дії або закривати ресурси

if exc\_type is not None:

print(f"An exception of type {exc\_type} occurred with value: {exc\_value}")

# Зазвичай тут повертаємо True, щоб вказати, що ми обробили виняток,

# і виконання продовжується. Якщо повернути False, виняток буде піднято наверх.

# Використання контекстного менеджера

with MyContextManager() as cm:

print("Inside the context")

# Можна виконувати операції в межах контексту

# Після виходу з контексту об'єкт контекстного менеджера очиститься

***10. Як поєднуються обробка виключних ситуацій і оператор with?***

**Приклад з обробкою виключень**:

class MyFileContextManager:

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

self.filename = filename

self.mode = mode

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

self.file = open(self.filename, self.mode)

return self.file

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

self.file.close()

if exc\_type is not None:

print(f"An exception of type {exc\_type} occurred with value: {exc\_value}")

# Можна вирішити, чи потрібно піднімати виняток наново або підавити його.

return False # Піднімати виняток наново

# Використання контекстного менеджера для роботи з файлом

try:

with MyFileContextManager("example.txt", "r") as file:

content = file.read()

# Якщо тут виникне помилка, вона буде оброблена у методі \_\_exit\_\_()

except FileNotFoundError:

print("File not found")

# Після виходу з контексту об'єкт файлу буде автоматично закритий

**Висновок:** оволодів навиками використання засобів мови Python для роботи з потоками і файлами.