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 5

Виконала:

ст. гр. КІ-305

Гринь С.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.

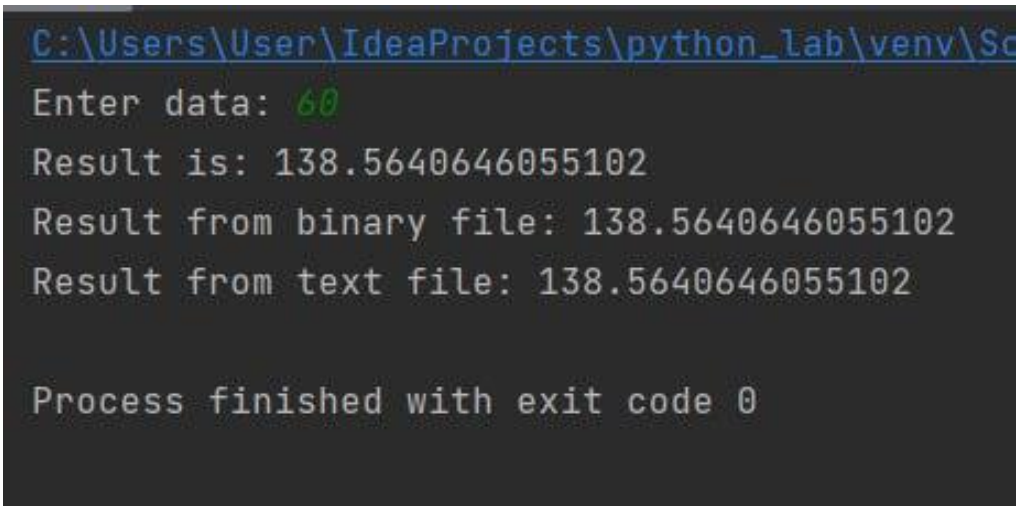
Варіант 5: y=2x/sin(x)

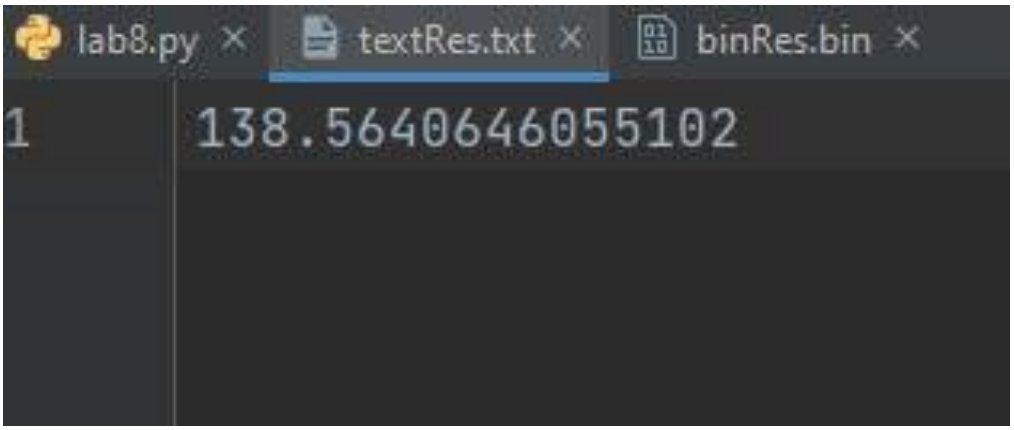
Код програми:

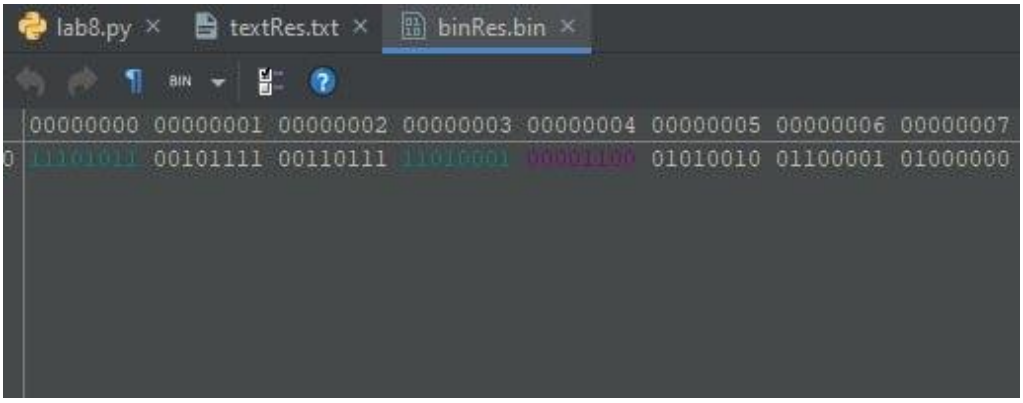
```
import os
import struct
import sys
import math

# Функція для запису результату у текстовий файл
def writeResTxt(fName, result):
    with open(fName, 'w') as f:
        f.write(str(result))

# Функція для зчитування результату з текстового файлу
def readResTxt(fName):
    result = 0.0
    try:
        if os.path.exists(fName):
            with open(fName, 'r') as f:
                result = float(f.read())
        else:
            raise FileNotFoundError(f"File {fName} not found.")
    except FileNotFoundError as e:
        print(e)
    return result

# Функція для запису результату у бінарний файл
def writeResBin(fName, result):
    with open(fName, 'wb') as f:
        f.write(struct.pack('d', result))

# Функція для зчитування результату з бінарного файлу
def readResBin(fName):
    result = 0.0
```

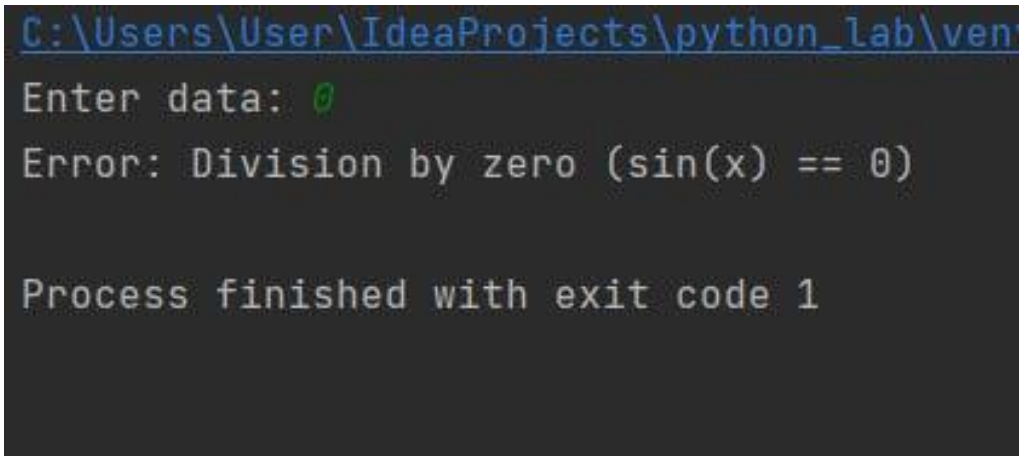
```
    try:
        if os.path.exists(fName):
            with open(fName, 'rb') as f:
                result = struct.unpack('d', f.read())[0]
        else:
            raise FileNotFoundError(f"File {fName} not found.")
    except FileNotFoundError as e:
        print(e)
    return result

# Функція для обчислення результату виразу
def calculate(x):
    try:
        rad = x * math.pi / 180.0  # Перетворення градусів в
радіани
        result = 2 * x / math.sin(rad)
        return result
    except ZeroDivisionError:
        print("Error: Division by zero (sin(x) == 0)")
        sys.exit(1)
    except ValueError:
        print("Error: Invalid input value for x")
        sys.exit(1)

if __name__ == "__main__":
    try:
        data = float(input("Enter data: "))  # Введення даних від
користувача
        result = calculate(data)
        print(f"Result is: {result}")

        # Запис результату в текстовий та бінарний файли
        writeResTxt("textRes.txt", result)
        writeResBin("binRes.bin", result)

        # Зчитування результату з бінарного файлу та текстового
файлу
        print("Result from binary file:
{0}".format(readResBin("binRes.bin")))
        print("Result from text file:
{0}".format(readResTxt("textRes.txt")))
    except ValueError as e:
        print(e)
        sys.exit(1)
    except FileNotFoundError as e:
        print(e)
        sys.exit(1)
```

Результат роботи програми:









Контрольні запитання

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

Мова Python має вбудований механізм обробки виключних ситуацій. Обробка виключних ситуацій забезпечується блоками try-except-finally.

Синтаксис:

```
try:
     <блок коду,  що може згенерувати виключення>
except <клас_виключення> as <посилання>:
     <блок коду обробника виключень>
else:
     <блок коду,  що виконується, якщо виключення не було
згенероване>
finally:
     <блок коду, який завжди виконується>
```

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:

```
# обробляє усі виключення. Доступу до об'єкту-виключення
немає.
except:

# обробляє виключення типу ValueError. Доступу до об'єкту-
виключення немає.
except ValueError:

# обробляє виключення типу RuntimeError, TypeError,
NameError. Доступу до об'єкту-виключення немає.
except (RuntimeError, TypeError, NameError):
```

Якщо `except` не містить жодного класу, то він буде реагувати на всі виключення, але доступитися до них буде неможливо через відсутність посилання на об'єкт класу-виключення.

Щоб доступитися до об'єкту-виключення слід вказати посилання на нього за допомогою наступного синтаксису:

```
except <клас-виключення> as <посилання>:
```

Приклад:

```
except ValueError as e:
```

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?

Ключовою функцією для роботи з файлами є функція `open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)`. Вона повертає дескриптор відкритого файлу або `None`. Параметри функції:

- `file` – шлях до файлу
- `mode` – режим відкривання файлу. Може приймати наступні значення та їх комбінації.

5. В яких режимах можна відкрити файл?

Таблиця 1.

Режими відкривання файлу

Параметр	Значення
'r'	Відкрити для читання (за замовчуванням)
'w'	Відкрити для запису, очистивши попередньо файл, якщо файл істує
'x'	Відкрити для ексклюзивного створення, якщо файл уже існує, то функція завершується невдачею
'a'	Відкрити для запису, дописуючи в кінець файлу, якщо він існує
'b'	Бінарний режим
't'	Текстовий режим (за замовчуванням)
'+'	Відкрити для оновлення (читання та запис)

6. Як здійснити читання і запис файлу?

Читання з файлів

Читання з файлів здійснюється за допомогою методу `read` об'єкту-файлу. Для читання однобайтних текстових рядків достатньо викликати метод `read` (для читання всього файлу чи певної кількості байт, кількість яких передається аргументом методу), або методу `readline` (для по-рядкового читання з файлу). Для читання даних інших типів вони мають бути записані як байтові послідовності, які вичитуються методом `read` після чого приводяться до відповідного типу. Для цього можна використати модуль `struct`, який призначений для полегшення інтерпретації байт як запакованих бінарних даних. Він перетворює значення Python на структури C, представлені як байтові об'єкти Python. Детальніше даний модуль описаний тут. Тож для розпаковування послідовності байт у певний тип слід використати метод `unpack` класу `struct`:

```
struct.unpack(format, buffer)
```

де `format` – визначає тип даних, які розпаковуються, а `buffer` – буфер, що містить послідовність байт, які треба розпакувати. Розмір буферу має відповідати типу даних, що розпаковується. Метод завжди повертає результат типу `tuple`, навіть якщо він містить лише одне значення. Тож наступний код вичитує одне бінарне число типу `double` з файлу і записує його у змінну `res`:

```
f = open('somefile.bin', 'wb')
res = struct.unpack('d', f.read())[0]
f.close()
```

Якщо файл має багато значень, а нам слід вичитати лише одне, тоді слід вказати кількість байт, що необхідно прочитати, наприклад,

```
f = open('somefile.bin', 'wb')
res = struct.unpack('d', f.read(struct.calcsize('d')))[0]
f.close()
```

Запис у файли

Запис у файл здійснюється за допомогою методу `write` об'єкту-файлу. Для запису однобайтних текстових рядків достатньо їх передати у метод `write`. При запису двійкових даних їх необхідно спочатку перетворити у послідовність байт. Для цього можна використати приведення до типу даних `bytearray`, метод `to_bytes` типу даних (наприклад, `int.to_bytes(var)`), або використати модуль `struct`. Тож, для запису бінарних даних їх спочатку треба запакувати у об'єкт, який являє собою послідовність байт та записати цю послідовність у файл. Для цього використаємо метод `pack` класу `struct`:

```
struct.pack(format, v1, v2, ...)
```

де `format` – визначає тип даних, які запаковуються, а значення `v1`, `v2`, ... - послідовність даних, які слід запакувати у бінарну структуру. Тож наступний код:

```
data = 2.0
res = struct.pack('d', data)
```

запакує дійсне число 2.0 у форматі подвійної точності (цей формат заданий аргументом `'d'`) у бінарну структуру, яка готова до зберігання у бінарному файлі:

```
data = 2.0
res = struct.pack('d', data)
f = open('somefile.bin', 'wb')
f.write(res)
f.close()
```

7. Особливості функцій у мові Python?

Функції у мові python не відрізняються за своєю суттю від функцій C/C++. Синтаксис оголошення функцій:

```
def function_name([параметри]):
  [оператори]
```

Приклади оголошення функцій:

```
def my_output():
  print("Hello world")

def my_output(txt1, txt2, delimitator = " "):
  """
     Outputs concatenated string
     :param txt1: the first text string
     :param txt2: the second text string
     :param delimitator: Delimitator. Space by default.
     :return: concatenated string
  """
  merged_text = delimitator.join((txt1, txt2))
  return merged_text
```

Приклади виклику функцій:

```
my_output("Hello ", "world")
my_output(txt1 = "Hello ", txt2 = "world") # Передача
параметрів за назвою параметра
```

8. Для чого призначенйи оператор with?

Оператор `with` використовується для автоматизації процесів закриття ресурсу і коректної обробки виключних ситуацій (аналог оператора try-з-ресурсами у Java). Наприклад, автоматичне закриття файлу, чи з'єднання після завершення роботи з ним, а також, при виникненні виключень. Таким ресурсом може бути будь-який об'єкт, клас якого містить визначені методи `__enter__` та `__exit__`, які дозволяють належним чином керувати ресурсами під час входу в блок `with`, виходу з нього та обробки виключних ситуацій. Такий об'єкт в термінах оператора `with` називається менеджером контексту. Детальний опис оператора `with` є у PEP-343.

Синтаксис:

```
with EXPRESSION as VAR:
    BLOCK
```

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: оволоділа навиками використання засобів мови Python для роботи з потоками і файлами.