Міністерство освіти і наук України

Національний університет “Львівська політехніка”

Кафедра ЕОМ

ФАЙЛИ ТА ВИКЛЮЧЕННЯ У PYTHON

Методичні вказівки
до лабораторної роботи № 8 з курсу “Кросплатформні засоби програмування”
для студентів базового напряму 6.123 - “Комп’ютерна інженерія”

Затверджено
на засіданні кафедри
”Електронні обчислювальні машини”
Протокол № 1 від 28.08.2023 р.

Львів – 2023

Файли та виключення у Python: Методичні вказівки до лабораторної роботи № 8 з курсу “Кросплатформні засоби програмування” для студентів базового напряму 6.123 - “Комп’ютерна інженерія” / Укладач: Олексів М.В. – Львів: Національний університет “Львівська політехніка”, 2023, 15 с.

Укладач Олексів М. В., к.т.н.

Рецензент

Відповідальний за випуск: Мельник А. О., професор, завідувач кафедри

ФАЙЛИ ТА ВИКЛЮЧЕННЯ У PYTHON

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

ТЕОРЕТИЧНІ ВІДОМОСТІ

Функції

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
```

Функції з довільної кількістю параметрів

У Python функції можуть мати довільну кількість параметрів. У цьому випадку їм можна передавати неіменовані або іменовані параметри, або їх комбінацію. Неіменовані параметри довільної кількості передаються за допомогою конструкції:

```
*args
```

При цьому всі передані аргументи поміщаються у артумент `args`, який є списком, що містить значення переданих аргументів.

Приклад оголошення функції з довільною кількістю неіменованих аргументів

```
def my_output(*args):
  """
     Outputs concatenated string
     :param args: the tuple of text strings
     :return: concatenated string
  """
  merged_text = str()
  for arg in args:
    merged_text = merged_text + " " + arg
  return merged_text
```

Приклад виклику функції з довільною кількістю неіменованих аргументів

```
 my_output("Hello ", "world")
```

Функції з довільною кількістю іменованих параметрів

Іншим способом передачі довільної кількості аргументів є використання іменованих параметрів, де при виклику аргументи передаються як пари значень – назва аргументу і його значення. При цьому використовується наступна конструкція:

```
**kwargs
```

Приклад оголошення функції з довільною кількістю іменованих аргументів

```
def my_output(**kwargs):
  """
     Outputs concatenated string
     :param kwargs: the dictionary of text strings
     :return: concatenated string
  """
  merged_text = str()
  for arg in kwargs.keys():
    merged_text = merged_text + " " + kwargs[arg]
  return merged_text
```

Приклад виклику функції з довільною кількістю іменованих аргументів

```
my_output(txt1 = "Hello", txt2 = "world", ... txtN = "ZZZZ")
```

Функція без тіла

Функція у Python не може зовсім не мати тіла. Якщо функція не має, тіла то слід вказати ключове слово `pass` як тіло такої функції.

```
def my_output(**kwargs):
  pass
```

Повернення кількох значень з функції

У Python можна передати з функції кілька результатів через оператор `return` через кому. У цьому випадку при виклику функції її результат треба буде присвоїти кільком змінним. Якщо функція повертає результат, який не використовується у програмі, то щоб уникнути оголошення змінної, яка ніде не буде використовуватися, застосовують символ "_" для таких результатів.

```
def few_output():
  return 1, 2
```

Приклад виклику функції

```
a, b = few_output() # a = 1, b=2
a = few_output() # a = (1, 2)
a, _ = few_output() # a = 1, другий результат ігноруємо
```

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>
```

Приклад:

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
```

```
except ValueError:
    print("Not a valid number.")
```

Блоки `try` є обов'язковим. Блок `try` містить код, який може згенерувати виключну ситуацію. 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`.

Файли

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.

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)

Параметри r,w,x,a можуть комбінуватися з параметрами b і t. Параметр + додається до інших параметрів за потреби.

- `buffering` — необов'язкове ціле число, яке використовується для встановлення політики буферизації. Значення 0 вимикає буферизацію (використовується лише в бінарному режимі); значення 1 активує буферизацію рядків (використовується лише в текстовому режимі); ціле число > 1, вкаує розмір у байтах буфера фрагментів фіксованого розміру. При значеннях 0 і 1 розмір буферу вказується окремо згідно повної документації цієї функції. Якщо параметр рівний -1, то у більшості випадків розмір буферу визначається автоматично.
- `encoding` – задає назву кодування тексту.
- `errors` – необов’язковий рядок, який визначає, як мають оброблятися помилки кодування та декодування. Не використовується в бінарному режимі.
- `newline` – визначає, як аналізувати символи нового рядка з потоку. Це може бути None, '', '\n', '\r' і '\r\n'.
- Якщо `closefd` має значення False і вказано дескриптор файлу, а не ім’я файлу, базовий дескриптор файлу залишатиметься відкритим, коли файл буде закрито. Якщо вказано назву файлу, то closefd має мати значення True (за замовчуванням); інакше виникне помилка.

- `opener` – посилання на користувацьку функцію, яка буде викликана для відкривання файлу, замість стандартної. Функція приймає параметри (file, flags) і повертає дескриптор відкритого файлу.

Приклад використання:

```
f = open('file-path', 'w', encoding="utf-8")
```

Після завершення роботи з файлом слід викликати метод `close` об'єкту-дескриптора файлу:

```
f.close()
```

Читання/запис відбувається за допомогою методів read/write та їх похідними об'єкту-дескриптора файлу.

Приклад читання з файлу:

```
fname = 'somefile.txt'

try:
     f = open(fname, 'r', encoding="utf-8")
     for line in f:
         print(line, end='')
except FileNotFoundError:
    print(f"File {fname} not found.  Aborting")
    sys.exit(1)
except OSError:
    print(f"OS error occurred trying to open {fname}")
    sys.exit(1)
except Exception as err:
    print(f"Unexpected error opening {fname} is",repr(err))
    sys.exit(1)
finally:
     f.close()
```

Приклад запису у файл:

```
text = 'text…'
fname = 'somefile.txt'

try:
     f = open('somefile.txt', 'w', encoding="utf-8")
     f.write(text)
```

```
except FileNotFoundError:
    print(f"File {fname} not found.  Aborting")
    sys.exit(1)
except OSError:
    print(f"OS error occurred trying to open {fname}")
    sys.exit(1)
except Exception as err:
    print(f"Unexpected error opening {fname} is",repr(err))
    sys.exit(1)
finally:
    f.close()
```

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()
```

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
```

де:

- EXPRESSION – вираз, який продукує об'єкт-менеджер контексту, або власне об'єкт-менеджер контексту;
- VAR – посилання на об'єкт, який перебуває під контролем менеджера контексту;
- BLOCK – блок коду, який має бути виконаний з використанням TARGET будучи обгорнутим при цьому конструкцією try-except-finally.

Конструкція "as VAR" є опціональною.
Ця конструкція семантично еквівалентна наступній:

```
manager = (EXPRESSION)
enter = type(manager).__enter__
exit = type(manager).__exit__
value = enter(manager)
hit_except = False

try:
    VAR = value
    BLOCK
except:
    hit_except = True
    if not exit(manager, *sys.exc_info()):
        raise
finally:
    if not hit_except:
        exit(manager, None, None, None)
```

Приклад читання з файлу з використанням оператора with:

```
fname = 'somefile.txt'

try:
     with open(fname, 'r', encoding="utf-8") as f:
     for line in f:
         print(line, end='')
except FileNotFoundError:
    print(f"File {fname} not found.  Aborting")
    sys.exit(1)
except OSError:
    print(f"OS error occurred trying to open {fname}")
    sys.exit(1)
except Exception as err:
    print(f"Unexpected error opening {fname} is",repr(err))
    sys.exit(1)
```

```
# Наступні операції виконуються оператором with автоматично
#finally:
#    f.close()
```

КОНТРОЛЬНІ ПИТАННЯ

1. За допомогою якої конструкції у мові Python обробляються виключні ситуації?
2. Особливості роботи блоку `except`?
3. Яка функція використовується для відкривання файлів у Python?
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`?
10. Як поєднуються обробка виключних ситуацій і оператор `with`?

ЛІТЕРАТУРА

1. Lutz M. Learning Python, 5th Edition / Mark Lutz. – O’Reilly, 2013. – 1643 p.
2. Васильєв О. Програмування мовою Python / Олексій Васильєв. – К: НК Богдан, 2019. – 504 с.
3. Python documentation [електронний ресурс]. – Режим доступу до документації: https://docs.python.org/3/
4. Python Enhancement Proposals [електронний ресурс]. – Режим доступу до документації: https://peps.python.org/

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.

ВАРІАНТИ ЗАВДАННЬ

1. y=tg(x)
2. y=ctg(x)
3. y=sin(x)/cos(x)
4. y=cos(x)/sin(x)
5. y=2x/sin(x)
6. y=tg(x)/sin(2x)
7. y=ctg(x)/sin(7x-1)
8. y=sin(x)/sin(2x-4)
9. y=tg(x)/3x
10. y=tg(x)/ctg(x)
11. y=ctg(x)/tg(x)
12. y=sin(x)/tg(4x)
13. y=sin(x)/ctg(8x)
14. y=cos(x)/tg(2x)
15. y=cos(2x)/ctg(3x-1)
16. y=7x/tg(2x-4)
17. y=(x-4)/sin(3x-1)
18. y=tg(x)/(sin(4x) – 2cos(x))
19. y=ctg(x)/(sin(2x) + 4cos(x))
20. y=tg(x)ctg(2x)
21. y=sin(3x-5)/ctg(2x)
22. y=tg(4x)/x
23. y=ctg(8x)/x
24. y=sin(x-9)/(x-cos(2x))
25. y=1/sin(x)
26. y=1/cos(4x)
27. y=1/tg(2x)
28. y=1/ctg(2x)
29. y=sin(x)/(x+tg(x))
30. y=cos(x)/(x+2ctg(x))

ПОРЯДОК ВИКОНАННЯ

1. Переконайтеся, що у вас є встановлені інтерпретатор Python і середовище для розробки Microsoft Visual Studio Code (VS Code).
2. Створіть папку для проекту на диску.
3. Відкрийте середовище VS Code.
4. Відкрийте вкладку Extensions (Ctrl + Shift + X) і переконайтеся, що у VS Code встановлено модуль розширення під назвою “Python”.
5. Відкрийте створену на кроці 2 папку, вибравши її з меню “File” -> “Open Folder”.
6. Створіть новий Python модуль (файл з розширенням .py), вибравши в меню “File” -> “New File” -> “Python file” і вказавши його ім’я.
7. Скопіюйте код демонстраційної програми, що наведена в методичних вказівках, у новостворений модуль і збережіть його.
8. Створіть віртуальне середовище Python, натиснувши Ctrl+Shift+P та вибравши зі списку команду “Python: Create Environment” -> “Venv” -> <шлях до встановленого інтерпретатора Python>. Після її виконання має появитися папка .venv у вашій робочій папці і вибраний інтерпретатор з віртуального середовища у нижньому правому куті рядка стану VS Code.
9. Дослідіть демонстраційну програму у відлагоджувачі.
10. Створіть власний модуль. Для цього повторіть крок 6 і збережіть новостворений модуль у робочій папці.
11. У створеному файлі напишіть програму згідно завдання.
12. Запустіть програму на виконання. Для цього слід вибрати підпункт меню “Run” -> “Start Without Debugging”.
13. Встановіть точки переривання, запустіть налагоджувач (“Run” -> “Start Debugging”) та покроково дослідіть процес виконання програми.

ДЕМОНСТРАЦІЙНА ПРОГРАМА

```
##############################################################################
# Copyright (c) 2023 Lviv Polytechnic National University. All Rights Reserved.
#
# This program and the accompanying materials are made available under the terms
# of the Academic Free License v. 3.0 which accompanies this distribution, and is
# available at https://opensource.org/license/afl-3-0-php/
#
# SPDX-License-Identifier: AFL-3.0
##############################################################################

import os
import struct
import sys

def writeResTxt(fName, result):
    with open(fName, 'w') as f:
        f.write(str(result))

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"File {fName} not found.")
    except FileNotFoundError as e:
        print(e)
    return result

def writeResBin(fName, result):
    with open(fName, 'wb') as f:
        # See https://docs.python.org/3/library/struct.html
        f.write(struct.pack('f', result))

def readResBin(fName):
    result = 0.0
    try:
        if os.path.exists(fName):
            with open(fName, 'rb') as f:
                # See https://docs.python.org/3/library/struct.html
                result = struct.unpack('f', f.read())[0]
        else:
            raise FileNotFoundError(f"File {fName} not found.")
    except FileNotFoundError as e:
        print(e)
    return result

def calculate(x):
        return x * x


if __name__ == "__main__":
    data = float(input("Enter data: "))
    result = calculate(data)
    print(f"Result is: {result}")
    try:
        writeResTxt("textRes.txt", result)
        writeResBin("binRes.bin", result)
        print("Result is: {0}".format(readResBin("binRes.bin")))
        print("Result is: {0}".format(readResTxt("textRes.txt")))
    except FileNotFoundError as e:
        print (e)
        sys.exit(1)
```

НАВЧАЛЬНЕ ВИДАННЯ

ФАЙЛИ ТА ВИКЛЮЧЕННЯ У PYTHON

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторної роботи № 8 з дисципліни “ Кросплатформні засоби програмування ”
для студентів базового напряму 6.123 - “Комп’ютерна інженерія”

Укладач Олексів М. В., к.т.н.

Редактор

Комп’ютерне верстання

Здано у видавництво . Підписано до друку
Формат 70x100/16. Папір офсетний. Друк на різографі
Умовн. друк. арк. Обл..-вид. арк..
Тираж прим. Зам..

Видавництво Національного університету “Львівська політехніка”
Реєстраційне свідоцтво ДК №751 від 27.12.2001 р.

Поліграфічний центр Видавництва
Національного університету “Львівська політехніка”

Вул. Ф. Колесси, 2. Львів, 79000