Лекція 24
Робота з файлами

Контрольна робота №8 (визначення варіанту)

ІО	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
61	7	8	1	2	3	4	5	8	7	2	3	1	5	6	4
62	4	5	6	1	2	3	4	5	6	7	8	5	1	2	6
63	3	6	5	8	1	2	3	4	5	6	7	4	3	1	8
64	2	7	4	5	6	1	2	3	4	5	6	8	7	3	5
65	1	2	3	4	5	6	1	2	3	4	5	6	7	8	3

Червоний колір – номер у списку групи

Чорний та синій колір – номер варіанту

ІО	16	17	18	19	20	21	22	23	24	25	26	27	28	29
61	1	8	2	3	4	5	6	1	2	3	4	5	6	7
62	8	7	6	5	2	3	4	5	6	1	2	3	4	5
63	6	5	8	4	1	2	3	4	5	6	7	1	2	3
64	4	2	5	7	8	3	1	2	3	4	5	6	7	1
65	2	3	4	6	5	7	8	3	1	2	3	7	5	4

1.Створіть клас **`Point`**, який задає координати точки на площині за умови, що початкові значення координат передаються як параметри при створенні екземпляра класу. Роздрукувати словник з наявними атрибути екземпляра класу з форматуванням.

2.Створіть клас **`Time`**, який задає два атрибути: години та хвилини. На друк виведіть значення атрибутів шляхом застосування оператора print до екземпляра класу з відповідними поясненнями та форматуванням. Роздрукуйте рядок документування

3.Створіть клас **`Person`**, який задає два атрибути: ім'я та вік людини. Виведіть на друк значення атрибутів шляхом виклику екземпляра класу **`man`** у такий спосіб: **`man(),`** з поясненнями та форматуванням. Роздрукуйте рядок документування.

4.Створіть клас **`Date`**, який задає два атрибути: **Y**-рік та **M**-місяць. Виведіть на друк значення атрибутів **Y** та **M** та повідомлення про відсутність атрибуту **D**, який задає число місяця, при спробі доступу до нього. Вивід сформатувати.

5.Створити клас **`Group`**, який містить атрибути: назва, факультет і курс. Даний клас має виводити повідомлення при доступі до довільного атрибуту класу з форматованим виводом імені атрибуту.

6.Створити клас **`Area`** без атрибутів. Записати в неіснуючий атрибут екземпляра класу значення периметру квадрата. Після запису атрибут повинен містити значення площі квадрата. Вивести на друк форматоване значення атрибута.

7.Створити підклас **`Me`** без атрибутів, який має два суперкласи: **`Dad`** та **`Mom`**. Суперкласи повинні мати атрибути з однаковим ім'ям **N.** Створити атрибут екземпляра класу **Me** також з ім'ям **N** та роздрукувати значення всіх атрибутів з форматуванням.

8.Створити клас **`Pupil`**. Створити атрибут екземпляра класу Pupil з ім'ям, яке складається з випадковим чином розміщених літер **`a`**,**`b`** і **`c`**. Потім перевірити наявність атрибуту з ім'ям **`abc`**. У випадку наявності такого атрибуту роздрукувати його значення, а за відсутності – роздрукувати словник з наявними атрибутами.

Відкриття файлу

Перш ніж працювати з файлом, необхідно створити об'єкт файлу за допомогою функції **open()** .

Функція має наступний формат:

```
open(<Шлях до файлу >[, mode='r'] [, buffering=-1] [, encoding=None] [,errors=None] [, newline=None] [, closefd=True]])
```

У першому параметрі вказують шлях до файлу. Шлях може бути абсолютним або відносним. Задаючи абсолютний шлях до файлу в Windows слід враховувати, що в Python слеш є спеціальним символом. Тому слеш необхідно подвоювати або замість звичайних рядків використовувати неформатовані рядки.

Абсолютний шлях до файлу

Приклад 1.

```
>>> "C:\\temp\\new\\file.txt" # Правильно
'C:\\temp\\new\\file.txt'
>>> r"C:\temp\new\file.txt" # Правильно
'C:\\temp\\new\\file.txt'
>>> "C:\temp\new\file.txt"  # Неправильно!!
'C:\temp\new\x0cile.txt'
```

Зверніть увагу на останній приклад. У цьому шляху через те, що слеші не подвоєні, виникла присутність відразу трьох спеціальних символів: `\t`, `\n` і `\f` (відображається як `\x0c`). Після перетворення цих спеціальних символів шлях буде мати вигляд:

```
C:<Табуляція>emp<Переведення рядка>ew
<Переведення формату>ile.txt
```

Виключення при доступі до файлу

Якщо в якості параметра функції `open()` передати помилковий рядок, то це призведе до виключення `OSError`:

```
>>> open("C:\temp\new\file.txt")

Traceback (most recent call last):
  File "<input>", line 1, in <module>
OSError: [Errno 22] Invalid argument:
'C:\temp\new\x0cile.txt'
```

Відносний шлях до файлу

Замість абсолютного шляху до файлу можна вказати відносний шлях. У цьому випадку шлях визначають з врахуванням розташування поточного робочого каталогу.

Поточним робочим каталогом будемо називати папку проекту.

Наприклад, якщо пака проекту розташована на шляху

C:\PYTHON,

то при задаванні відносного шляху

folder1\folder2\file.txt

повний шлях до файлу матиме вигляд:

C:\PYTHON\ folder1\folder2\file.txt

Функція abspath()

Відносний шлях можна перетворити в абсолютний шлях за допомогою функції **abspath ()** з модуля **os.path**.

Можливі наступні варіанти:

1. Файл в поточному робочому каталозі. Можна вказати тільки назву файлу.

Приклад 2.

```
>>> import os.path # Підключаємо модуль

>>> #файл у поточному робочому каталозі (C:\PYTHON\)

>>> os.path.abspath(r"file.txt")

'C:\\PYTHON\\file.txt'
```

2.Файл у вкладеній папці. Перед назвою файлу потрібно вказати назви вкладених папок через слеш.

Приклад 3.

```
>>>import os.path
>>>  # файл, що відкриваємо, C:\PYTHON\folder1\

>>> os.path.abspath(r"folder1\file.txt")

'C:\\PYTHON\\folder1\\file.txt'

>>>  # файл, що відкриваємо,в C:\PYTHON\folder1\folder2\

>>>  os.path.abspath(r"folder1\folder2\file.txt")

'C:\\PYTHON\\folder1\\folder2\\file.txt'
```

3.Файл на рівень вище. Перед назвою файлу вказують дві крапки й слеш ("..\").

Приклад 4.

```
>>>import os.path
>>> # файл, що відкриваємо, на рівень вище

>>> os.path.abspath(r" ..\file.txt")

'C:\\PYTHON\\ .. \\file.txt'

>>> # файл, що відкриваємо, на 2 рівня вище

>>> os.path.abspath(r" ....\file.txt")

'C:\\PYTHON\\ .... \\file.txt'
```

4. На початку шляху розташований слеш. Шлях будують від кореня диска. У цьому випадку місце розташування поточного робочого каталогу не має значення.

Приклад 5.

```
>>>import os.path
>>># файл, що відкриваємо, в C:\ PYTHON \folder1\

>>> os.path.abspath(r"\ PYTHON \folder1\file.txt")
'C:\\ PYTHON \\folder1\\file.txt'


>>>#файл, що відкриваємо,в  C:\ PYTHON \folder1\folder2\
>>> os.path.abspath(r"\PYTHON\folder1\folder2\file.txt")
'C:\\PYTHON\\folder1\\folder2\\file.txt'
```

Прямі та зворотні слеші

В абсолютному й відносному шляхах допустимо використання як прямих, так і зворотних слешів. Усі вони будуть автоматично перетворені з урахуванням значення атрибута `sep` з модуля `os. path`. Значення цього атрибута залежить від використовуваної операційної системи. Виведемо значення атрибута `sep` в операційній системі Windows:

Приклад 6.

```
>>>import os.path

>>> os.path.sep
'\\'
>>> os.path.abspath(r"C:/PYTHON/folderl/file.txt")
'C:\\PYTHON\\folderl\\file.txt'
```

Особливості використання відносного шляху

При використанні відносного шляху необхідно враховувати місце розташування поточного робочого каталогу.

Робочий каталог не завжди збігається з каталогом, у якому перебуває файл, що виконується.

Якщо файл запускається за допомогою подвійного клацання на його значку, то каталоги будуть збігатися.

Якщо ж файл запускається з командного рядка, то поточним робочим каталогом буде каталог, в якому ми знаходимося під час запуску файлу.

В каталозі `C:\PYTHON` створимо наступну структуру файлів:

```
C:\PYTHON\
    first.py
    folder1\
        __init__.py
        module1.py
```

Приклад 6. Вміст файлу `C:\PYTHON\first.py`

```
import os, sys
print("%-25s%s" % ("Файл:", os.path.abspath(__file__)))
print("%-25s%s" % ("Поточний робочий каталог:", os.getcwd()))
print("%-25s%s" % ("Каталог для завантаження:", sys.path[0]))
print("%-25s%s" % ("Шлях до файлу:", os.path.abspath("file.txt")))
print("-" * 40)
import folder1.module1 as m
m.get_cwd ()
```

Файл `C:\PYTHON\folder1\_init_.py` створюємо порожнім. Як ви вже знаєте, цей файл указує інтерпретаторові Python, що даний каталог є пакетом з модулями.

Вміст файлу `C:\PYTHON\folder1\module1.py` наведений в прикладі.

Приклад 7.

```
import os, sys
def get_cwd():
    print("%-25s%s" % ("Файл:", os.path.abspath(__file__ )))
    print("%-25s%s" % ("Поточний робочий каталог:", os.getcwd()))
    print("%-25s%s" % ("Каталог для завантаження:", sys.path[0]))
    print ("%-25s%s" % ("Шлях до файлу:", os.path.abspath("file.txt")))
```

Запускаємо командний рядок, переходимо в каталог `C:\PYTHON` і запускаємо файл **`first.py`**:

```
Файл:                          C:\PYTHON\ first.py
Поточний робочий каталог: C:\PYTHON
Каталог для завантаження: C:\PYTHON
Шлях до файлу:                 C:\PYTHON\file.txt
-----------------------------------
Файл:     C:\PYTHON\folder1\module1.py
Поточний робочий каталог: C:\PYTHON
Каталог для завантаження: C:\PYTHON
Шлях до файлу:                 C:\PYTHON\file.txt
```

У цьому прикладі поточний робочий каталог збігається з каталогом, у якому розташований файл `first.py`.

Однак зверніть увагу на поточний робочий каталог всередині модуля `module1.py`.

Якщо всередині цього модуля у функції `open ()` указати назву файлу без шляху, то пошук файлу буде зроблений у каталозі `C:\PYTHON`, а не `C:\PYTHON \folder1`.

Тепер перейдемо в корінь диска `C:` і знову запустимо файл `first.py`:

```
C:\>PYTHON\first.py

Файл:                          C:\PYTHON\ first.py
Поточний робочий каталог:C:\
Каталог для завантаження:C:\PYTHON
Шлях до файлу:                 C:\file.txt
-------------------------------------
Файл:              C:\PYTHON\folder1\module1.py
Поточний робочий каталог:  C:\
Каталог для завантаження:  C:\PYTHON
Шлях до файлу:              C:\file.txt
```

У цьому випадку поточний робочий каталог не збігається з каталогом, у якому розташований файл `first.py`.

Якщо усередині файлів `first.py` і `module1.py` у функції `open()` указати назву файлу без шляху, то пошук файлу буде проводитися в корені диску `C:`, а не в каталогах із цими файлами.

Щоб пошук файлу завжди проводився в каталозі з файлом, що виконується, необхідно цей каталог зробити поточним за допомогою функції **`chdir()`** з модуля `os`.

Для прикладу змінимо вміст файлу `first.py`

Приклад 8.

```
import os, sys
# Робимо каталог з файлом, що виконується, поточним
os.chdir(os.path.dirname (os.path.abspath(__file__)))
print("%-25s%s" % ("Файл:", __file__ ) )
print("%-25s%s" % ("Поточний робочий каталог:", os.getcwd()))
print("%-25s%s" % ("Каталог для завантаження:", sys.path[0]))
print("%-25s%s" % ("Шлях до файлу:", os.path.abspath("file.txt")))
```

Зверніть увагу на червоний рядок. За допомогою атрибута `__file__` ми одержуємо шлях до файлу, що виконується, разом з назвою файлу.

Атрибут `__file__` не завжди містить повний шлях до файлу.

Щоб завжди одержувати повний шлях до файлу, слід передати значення атрибута у функцію `abspath()` з модуля `os.path`.

Далі ми «витягаємо» шлях (без назви файлу) за допомогою функції `dirname()` і передаємо його функції `chdir()`.

Тепер, якщо у функції `open ()` указати назву файлу без шляху, то пошук буде проводитися в каталозі з цим файлом.

Запустимо файл `first.py` із `C:>\PYTHON\first.py`

```
Файл:                          C:/PYTHON/first.py
Поточний робочий каталог:      C:\PYTHON
Каталог для імпорту:           C:\PYTHON
Шлях до файлу:                 C:\PYTHON\file.txt
```

Той же результат одержимо при запуску файлу із `C:\`

Основний висновок

1. Поточним робочим каталогом буде каталог, з якого запускається файл, а не каталог, у якому розташований файл, що виконується.

2. Шляхи пошуку файлів не мають жодного відношення до шляхів пошуку модулів.

Параметр mode у функції open()

```
open(<Шлях до файлу>[, mode='r'] [, buffering=-1] [, encoding=None] [,errors=None] [, newline=None] [, closefd=True]])
```

Необов'язковий параметр **mode** у функції **open()** може набувати наступних значень:

r – тільки читання (значення за замовчуванням).
Після відкриття файлу покажчик установлюється на початок файлу. Якщо файл не існує, виконується виключення `Filenotfounderror;`

r+ – читання й запис.
Після відкриття файлу покажчик установлюється на початок файлу. Якщо файл не існує, то виконується виключення `Filenotfounderror;`

w – запис.

Якщо файл не існує, він буде створений. Якщо файл існує, він буде перезаписаний. Після відкриття файлу покажчик установлюється на початок файлу;

w+ – читання й запис.

Якщо файл не існує, він буде створений. Якщо файл існує, він буде перезаписаний. Після відкриття файлу покажчик установлюється на початок файлу;

a – запис.

Якщо файл не існує, він буде створений. Запис здійснюється в кінець файлу. Вміст файлу не видаляється;

a+ – читання й запис.

Якщо файл не існує, він буде створений. Запис здійснюється в кінець файлу. Вміст файлу не видаляється;

x – створення файлу для запису.

Якщо файл уже існує, виконується виключення `Fileexistserror`;

x+ – створення файлу для читання й запису.

Якщо файл уже існує, виконується виключення `Fileexistserror`.

Модифікатори режиму (читання - запис)

Після вказівки режиму може слідувати модифікатор:

`b` – файл буде відкритий у бінарному режимі.

Файлові методи приймають і повертають об'єкти типу `bytes`;

`t` – файл буде відкритий у текстовому режимі (значення за замовчуванням у Windows).

Файлові методи запису та считування рядків

Файлові методи обробляють об'єкти типу `str`.

У цьому режимі буде автоматично виконуватися обробка символу кінця рядка.

Для прикладу створимо файл `file.txt` і запишемо в нього два рядки:

Приклад 9.

```
>>> f = open(r"file.txt","w")# Відкриваємо файл на запис
>>>  f.write("String1\nstring2")# Записуємо два рядки у файл
15
>>> f.close()# Закриваємо файл
```

Оскільки ми вказали режим `w`, то якщо файл не існує, він буде створений, а якщо існує, то буде перезаписаний.

Тепер виведемо вміст файлу в бінарному й текстовому режимах:

Приклад 10.

```
>>> # Бінарний режим (символ \r залишається)
>>> with open(r"file.txt", "rb") as f:
        for line in f:
            print(repr(line))

b'string1\r\n'
b'string2'

>>> #Текстовий режим (символ \r видаляється)
>>> with open(r"file.txt", "r") as f:
        for line in f:
           print(repr(line))

'String1\n'
'String2'
```

Буферизація запису в файл (параметр buffering)

```
open(<Шлях до файлу>[, mode='r'] [, buffering=-1] [, encoding=None] [,errors=None] [, newline=None] [, closefd=True]])
```

Для прискорення роботи проводиться буферизація записуваних даних.

Інформація з буфера записується у файл повністю тільки в момент закриття файлу або після виклику функції або методу `flush()`.

Керування буферизацією відбувається через параметр `buffering`.

У необов'язковому параметрі `buffering` можна вказати розмір буфера.

`buffering = 0`, то дані будуть відразу записуватися у файл (значення припустиме тільки в бінарному режимі).

`buffering =1` використовується при по-рядковому записі у файл (значення припустиме тільки в текстовому режимі),

`buffering =N`, де N - додатне число

Задає приблизний розмір буфера,

`buffering = -N` від'ємне значення (або відсутність значення)

Задає установку розміру, застосовуваного в системі за замовчуванням. За замовчуванням текстові файли буферизуются по-рядково, а бінарні – частинами, розмір яких інтерпретатор вибирає самостійно в діапазоні від 4096 до 8192 байтів.

Кодування запису в файл (параметр encoding)

open (<**Шлях до файлу** >[, mode='r'] [, buffering=-1] [, encoding=None] [,errors =None] [, newline=None] [, closefd=True]**)**

При використанні текстового режиму кодування задається за замовчуванням.

При читанні проводиться спроба перетворити дані в кодування Unicode.

При записі виконується зворотна операція – рядок перетвориться в послідовність байтів у визначеному кодуванні.

За замовчуванням призначається кодування, застосовуване в системі. Якщо перетворення неможливе, то виконується виключення. Указати кодування, яке буде використовуватися при записі й читанні файлу, дозволяє параметр **encoding.**

Для прикладу запишемо дані в кодуванні UTF-8:

Приклад 11.

```
>>> f = open(r"file.txt", "w", encoding="utf-8")
>>> f.write("Рядок") # Записуємо рядок у файл
>>> f.close() # Закриваємо файл
```

Для читання цього файлу слід явно вказати кодування при відкритті файлу:

Приклад 12.

```
>>> with open(r"file.txt", "r", encoding="utf-8") as f:
        for line in f:
            print(line)
Рядок
```

Особливості роботи з кодуванням UTF

При роботі з файлами в кодуваннях UTF-8, UTF-16 і UTF-32 слід враховувати, що на початку файлу можуть бути наявними службові символи, називані скорочено BOM (Byte Order Mark, мітка порядку байтів).

Для кодування UTF-8 ці символи є необов'язковими, і в попередньому прикладі вони не були додані у файл при записі.

Щоб символи були додані, у параметрі `encoding` слід указати значення `utf-8-sig`. Запишемо рядок у файл у кодуванні UTF-8 з BOM:

Приклад 13.

```
>>> f = open (r"file.txt", "w", encoding="utf-8-sig")
>>> f.write("Рядок") # Записуємо рядок у файл
>>> f.close () # Закриваємо файл
```

Прочитаємо файл з різними значеннями в параметрі `encoding`:

Приклад 14.

```
>>> with open(r"file.txt", "r", encoding="utf-8") as f:
        for line in f:
            print(repr(line))
'\ufeffРядок'
>>> with open(r"file.txt", "r", encoding="utf-8-sig") as f:
        for line in f:
            print(repr(line))
'Рядок'
```

У першому прикладі ми вказали значення utf-8, тому маркер БОМ був прочитаний з файлу разом з даними.

У другому прикладі вказано значення utf-8-sig, тому маркер БОМ не потрапив в результат.

Якщо невідомо, чи є маркер у файлі, і необхідно отримати дані без маркера, то слід завжди вказувати значення utf-8-sig при читанні файлу в кодуванні UTF-8.

Для кодувань UTF-16 і UTF-32 маркер БОМ є обов'язковим. При вказівці значень utf-16 і utf-32 в параметрі encoding обробка маркера проводиться автоматично.

При запису даних маркер автоматично вставляється в початок файлу, а при читанні він не потрапляє в результат.

Запишемо рядок в файл, а потім прочитаємо його з файлу:

Приклад 15.

```
>>> with open(r"file.txt", "w", encoding = "utf-16") as f:
        f.write ( "Рядок")
5
>>> with open (r"file.txt", "r", encoding = "utf-16") as f:
            for line in f:
                print (repr (line))
'Рядок'
```

Параметр error

```
open(<Шлях до файлу>[, mode='r'] [, buffering=-1] [, encoding=None] [,errors=None] [, newline=None] [, closefd=True]])
```

У параметрі **errors** можна вказати рівень обробки помилок. Можливі значення:

"strict" (при помилці виконується виключення Valueerror – значення за замовчуванням),
"replace" (невідомий символ заміняється символом знак питання або символом з кодом \ufffd),
"ignore" (невідомі символи ігноруються),
"xmlcharrefreplace" (невідомий символ заміняється послідовністю &#xxxx;) і
"backslashreplace" (невідомий символ заміняється послідовністю \uxxxx).

Параметр newline

open (<**Шлях до файлу** >[, mode='r'] [, buffering=-1] [, encoding=None] [,errors =None] [, newline=None] [, closefd=True]**)**

Параметр newline задає режим обробки символів кінця рядків.

Підтримувані ним значення такі:

- None (значення за замовчуванням) – виконується стандартна обробка символів кінця рядка. Наприклад, в Windows при читанні символи \r\n перетворяться в символ \n, а при записі проводиться зворотне перетворення;
- "" (порожній рядок) – обробка символів кінця рядка не виконується;
- "<Спеціальний символ>" – зазначений спеціальний символ використовується для позначення кінця рядка, і ніяка додаткова обробка не виконується. Як спеціальний символ можна вказати лише \r\n, \r і \n.