Лекція 25

Робота з файлами (продовження)

Методи для роботи з файлами

Після відкриття файлу функція `open()` повертає об'єкт, за допомогою якого проводиться подальша робота з файлом.

Тип об'єкта залежить від режиму відкриття файлу й буферизации. Розглянемо основні методи:

- `close()` – закриває файл.

Оскільки інтерпретатор автоматично видаляє об'єкт, коли на нього відсутні посилання, у невеликих програмах можна явно не закривати файл.

Проте, явне закриття файлу є ознакою гарного стилю програм. Мова Python підтримує протокол менеджерів контексту.

With..as

Використання with..as гарантує закриття файлу незалежно від того, відбулося виключення всередині блоку коду чи ні.

Приклад 1

```
with open(r"file.txt", "w", encoding="cp1251") as f:
        f.write("Рядок")
# Записуємо рядок у файл
# Тут файл уже закритий автоматично
```

Метод write

- `f.write (<дані>)` – записує рядок або послідовність байтів у файл.

Якщо як параметр зазначений рядок, то файл повинен бути відкритий у текстовому режимі.

Для запису послідовності байтів необхідно відкрити файл у бінарному режимі.

Пом’ятайте, що не можна

записувати рядок у бінарному режимі та
послідовність байтів у текстовому режимі.

Метод повертає кількість записаних символів або байтів.

Приклад 2. Приклад запису у файл

```
# Текстовий режим
f = open(r"file.txt", "w",encoding="cp1251")

f.write("Рядок1\nрядок2")#Записуємо рядок у файл

f.close()

f = open(r"file.txt", "wb")

#f.write("Рядок1\nрядок2")# не пишіть рядки в
бінарний файл
f.write(bytes("Рядок1\nрядок2", "cp1251"))


f.write(bytearray("\nрядок3", "cp1251"))


f.close()
```

Метод writelines

- `writelines (<Послідовність>)` – записує послідовність у файл.

Послідовність складається з рядків

Якщо всі елементи послідовності є рядками, то файл повинен бути відкритий у текстовому режимі.

Послідовність складається з байтів

Якщо всі елементи є послідовностями байтів, то файл повинен бути відкритий у бінарному режимі.

Приклад 3. Приклад запису елементів списку:

```
>>> # Текстовий режим

>>> f = open(r"file.txt", "w+", encoding="cp1251")
>>> f.writelines(["Рядок1\n", "Рядок2"])
>>> f.close()


>>> # Бінарний режим

>>> f = open(r"file.txt", "wb")
>>>arr = [bytes("Рядок1\n", "cp1251"), bytes("Рядок2", "cp1251")]
>>> f.writelines(arr)
>>> f.close()
```

Метод writable

- `writable()` – повертає `True або False`

Якщо файл підтримує запис - `True`,

Якщо не підтримує запис –`False`

Приклад 4

```
>>> f = open(r"file.txt", "r")
>>> f.writable()
False

>>> f = open(r"file.txt", "w")
>>> f.writable()
True
```

Метод read

- `read([<Кількість>])` – зчитує дані з файлу.

Якщо файл відкритий у текстовому режимі, то повертається рядок.

Якщо файл відкритий у бінарному режимі, то повертається послідовність байтів.

Якщо параметр не зазначений, то повертається вміст файлу від поточної позиції покажчика до кінця файлу:

Приклад 5

```
# Текстовий режим
with open(r"file.txt","r", encoding="cp1251") as f:
    f.read()
Рядок1
рядок2
# Бінарний режим
with open(r"file.txt", "rb") as f:
    f.read()
b'\xd0\xff\xe4\xee\xea1\n\xd0\xff\xe4\xee\xea2'
```

Метод read з параметром

Якщо в методі read як параметр вказати число, то за кожний виклик буде повертатися зазначена кількість символів або байтів. Коли досягнуто кінець файлу, метод повертає порожній рядок.

Приклад 6

```
# Текстовий режим
with open(r"file.txt", "r") as f:
        A=f.read(6)
        B=f.read(7)
        C=f.read(7)
print(A, B, C)
```

Результат роботи:

```
Рядок1
рядок2
```

Метод readline()

`readline([<Кількість>])` – зчитує з файлу один рядок при кожному виклику.

Якщо файл відкритий у текстовому режимі, то повертається рядок, а якщо в бінарному – послідовність байтів.
Рядок, що повертається, включає символ переводу рядка.

Виключенням є останній рядок – якщо він не завершується символом переводу рядка, то він доданий не буде.

При досягненні кінця файлу повертається порожній рядок.

Приклад 7.

```
# Текстовий режим
f = open(r"file.txt","r",encoding="cp1251")
print(f.readline(),f.readline())
print(f.readline())
f.close()

Рядок1
 Рядок2

# Бінарний режим
f = open(r"file.txt", "rb")
print(f.readline(),f.readline())
print(f.readline())#Досягнутий кінець файлу
f.close()
b'\xd0\xff\xe4\xee\xea1\r\n' b'\xd0\xff\xe4\xee\xea2'
b''
```

Метод readline() з параметром

Якщо в параметрі зазначене число, то зчитування буде виконуватися доти, поки не зустрінеться:

- символ нового рядка (\n),
- символ кінця файлу,
- з файлу не буде прочитана зазначена кількість символів.

Якщо кількість символів у рядку менша за значену в параметрі, то буде прочитаний один рядок, а не зазначена кількість символів,

```
Symbol\n – 6 символів readline(8)
```

Якщо кількість символів у рядку більша, то повертається зазначена кількість символів.

```
Symbol\n – 6 символів readline(3)
```

Приклад 8.

```
f = open (r"file.txt", "w", encoding="cp1251")
f.write("Рядок1\nрядок2")
f.close()

f = open (r"file. txt", "r", encoding="cp1251")
p = f.readline(2); print(p)
p = f.readline(2); print(p)
p = f.readline(100); print(p)
```

Результат виконання:

Ря

до

к1

Метод readlines()

`readlines()` – зчитує весь вміст файлу в список.

Кожний елемент списку буде містити один рядок, включаючи символ переводу рядка.

Виключенням є останній рядок. Якщо він не завершується символом переводу рядка, то символ переводу рядка доданий не буде.

Якщо файл відкритий у текстовому режимі, то повертається список рядків, а якщо в бінарному – список об'єктів типу `bytes`.

Приклад 9.

```
>>>#Текстовый режим
f = open (r"file.txt", "w", encoding="cp1251")
f.write("Рядок1\nРядок2")
f.close()
f = open (r"file. txt", "r", encoding="cp1251")
print(f.readlines())
```

Результат виконання:

```
['Рядок1\n', 'Рядок2']
```

```
>>> # Бинарный режим
with open (r"file. txt", "rb") as f:
    print(f.readlines())
```

Результат виконання:

```
[b'\xd0\xff\xe4\xee\xea1\r\n', b'\xd0\xff\xe4\xee\xea2']
```

Метод `__next__ ()`

`__next__ ()` – зчитує один рядок при кожному виклику.

Якщо файл відкритий у текстовому режимі, повертається рядок,

а якщо в бінарному – послідовність байтів.

При досягненні кінця файлу виконується виключення `Stopiteration`.

Приклад 10.

```
#Текстовий режим
f = open(r"file.txt", "r",
encoding="cp1251")
print(f.__next__())
print(f.__next__())
print(f.__next__())
f.close()
Результат выполнения:
Рядок1
Рядок2

Traceback (most recent call last):

  File "C:/PYTHON/first.py", line 5, in <module>
    print(f.__next__()) # Досягнутий кінець файлу
StopIteration
```

Перебір файлу по рядках: `__next__()` та `for`

Завдяки методу `__next__()` ми можемо перебирати файл построчно за допомогою циклу **`for`**.

Цикл `for` на кожній ітерації буде автоматично викликати метод `__next__()`. Для прикладу виведемо всі рядки, попередньо вилучивши символ переводу рядка:

Приклад 11.

```
f = open(r"file.txt", "r",encoding="cp1251")
for line in f:print(line.rstrip("\n"),end=" ")
f.close ()
```

Результат виконання:

Рядок1 Рядок2

Методи flush() та fileno()

`flush()` – примусово записує дані з буфера на диск;
`fileno()` – повертає цілочисельний дескриптор файлу.

Значення, що повертається завжди буде більшим за число 2.

Число 0 закріплене за стандартним введенням `stdin`,
Число 1 – за стандартним виводом `stdout`,
Число 2 – за стандартним виводом повідомлень про помилки `stderr`.

Приклад 12.

```
f = open(r"file.txt","r", encoding="cp1251")
print("Дескриптор =",f.fileno ())
f.close ()
Результат виконання:
Дескриптор = 3
```

Метод truncate ([<Кількість>])

truncate ([<Кількість>]) – обрізає файл до зазначеної кількості символів (якщо заданий текстовий режим)

зазначеної кількості байтів (у випадку бінарного режиму).

Метод повертає новий розмір файлу.

Приклад 13.

```
#Записуємо в файл
f = open (r"file.txt", "w", encoding="cp1251")
f.write("Рядок1\nРядок2")
f.close()
```

```
# Зчитуємо з файлу
f = open (r"file.txt", "r+", encoding="cp1251")
print(f.read())
f.truncate(5)
f .close ()

print("Після урізання отримали")
with open(r"file.txt", "r", encoding="cp1251") as f:
   print(f. read ())
```

Результат виконання:

Рядок1

Рядок2

Після урізання отримали

Рядок

Метод tell()

`tell()` – повертає позицію покажчика відносно початку файлу у вигляді цілого числа.

`\r` - додатковий байт, хоча цей символ видаляється при відкритті файлу в текстовому режимі.

Приклад 14.

```
with open(r"file.txt", "w", encoding="cp1251") as f:
    f.write("String1\nString2")
f = open(r"file.txt", "r", encoding="cp1251")
print("Поточна позиція:",f.tell()) # Покажчик на початку файлу
print(f.readline()) # Перемещуємо покажчик
print("Поточна позиція:",f.tell()) # Повертає 9 (8 + '\r'), а не 8 '•'
f.close()
```

Результат виконання:

```
Поточна позиція: 0
String1
Поточна позиція: 9
```

Як позбавитися невідповідності з\r

Щоб уникнути даної невідповідності, слід відкривати файл у бінарному режимі, а не в текстовому:

Приклад 15.

```
f = open(r"file.txt", "rb")
print(f.readline()) # Переміщуємо покажчик
print("Поточна позиція:",f.tell())
f.close()

Результат виконання:
b'String1\r\n'
Поточна позиція: 9
```

Метод seek(<Зсув>[, <Позиція>])

seek(<Зсув>[,<Позиція>]) – установлює покажчик у позицію, що має зсув <Зсув> відносно позиції <Позиція>.

У параметрі <Позиція> можуть бути зазначені наступні атрибути з модуля io або відповідні їм значення:

- io.SEEK_SET або 0 – початок файлу (значення за замовчуванням);
- io.SEEK_CUR або 1– поточна позиція покажчика. Додатне значення зсуву викликає переміщення у напрямку кінця файлу, від'ємне – у напрямку початку;
- io.SEEK_END або 2 – переміщення від кінця файлу.

Виведемо значення цих атрибутів:

```
>>> import io
>>> io.SEEK_SET, io.SEEK_CUR, io.SEEK_END
(0, 1, 2)
```

Приклад використання методу `seek()` :

Приклад 16.

```
import io
f = open(r"file.txt", "rb")
print("Зсунути на ",f.seek(9, io.SEEK_CUR)) # 9 байт від покажчика
print("Поточна позиція:",f.tell())
print("На початок: ",f.seek(0,io.SEEK_SET)) # Зсуваємо на початок
print("Поточна позиція:",f.tell())
print("-9 від кінця",f .seek(-9, io. SEEK_END)) # -9 байтов від кінця
print("Поточна позиція:",f.tell())
f .close ()
```

Результат виконання:

```
Зсунути на  9
Поточна позиція: 9
На початок:  0
Поточна позиція: 0
-9 від кінця 7
Поточна позиція: 7
```

Метод seekable()

`seekable()` – повертає `True`, якщо покажчик файлу можна зсунути в іншу позицію, і `False` – якщо ні:

Приклад 17.

```
f = open(r"file.txt", "r")
print(f. seekable ())
Результат виконання: True
```

Крім методів, об'єкти файлів підтримують кілька атрибутів:

`name` – ім'я файлу;

`mode` – режим, у якому був відкритий файл;

`closed` – повертає `True`, якщо файл був закритий, і `False` – якщо ні.

Приклад застосування атрибутів файлу name, mode і closed

Приклад 18.

```
f = open(r"file.txt", "rb+")
print("Им'я файлу:",f.name)
print("Режим відкривания:",f.mode)
print("До закриття",f.closed)
f.close()
print("Після закриття",f.closed)
```

```
Результат виконання:
Ім'я файлу: file.txt
Режим відкривання: rb+
До закриття False
Після закриття True
```

Атрибут `encoding`

`encoding`–назва кодування, яке буде використовуватися для перетворення рядків перед записом у файл або при читанні. Атрибут доступний тільки в текстовому режимі.

Змінити значення атрибута не можна, оскільки він доступний тільки для читання.

Приклад 19.

```
f = open(r"file.txt", "a", encoding="cp1251")
print("Кодування: ",f.encoding)
f.close()
```

Результат виконання: Кодування: cp1251

Застосування encoding у stdout

Стандартний вивід `stdout` також є файловим об'єктом.

Атрибут `encoding` цього об'єкта завжди містить кодування пристрою виводу, тому рядок перетвориться в послідовність байтів у правильному кодуванні.

Наприклад, при запуску у вікні редактора Pycharm – значення `"UTF-8"`.

Приклад 20.

```
import sys
# Кодування фала стандартного виводу stdout
print("Кодування STDOUT =",sys.stdout.encoding)
```

Атрибут `buffer`

`buffer` – дозволяє одержати доступ до буфера.

Атрибут доступний тільки в текстовому режимі.

За допомогою цього об'єкта можна записати послідовність байтів у текстовий потік.

Приклад 21.

```
#Приклад застосування атрибута buffer
f = open(r"file.txt", "w", encoding="cp1251")
print(f.buffer.write(bytes("Рядок","cp1251")))
#Повертає кількість символів
f.close()
```

Функції для маніпулювання файлами
shutil

Для копіювання й переміщення файлів призначені наступні функції з модуля `shutil`:

`copyfile``(<Початковий файл>,<Куди копіюємо>)`

Функція дозволяє скопіювати вміст файлу в інший файл.

Ніякі метадані (наприклад, права доступу) не копіюються.

Якщо файл існує, то він буде перезаписаний.

Якщо файл не вдалося скопіювати, виконується виключення `OsError` або одне з виключень, що є підкласом цього класу.

Приклад 22.

```
import shutil # Підключаємо модуль
shutil.copyfile(r"file.txt", r"file2.txt")
Результат виконання:
У папці C:\PYTHON з'являється файл file2.txt
import shutil #шлях не існує:
shutil.copyfile(r"file.txt", r"C:\P_NEW\file2.txt")
 Результат виконання:
   .......
   FileNotFoundError: [Errno 2] No such file or directory: 'C:\\P_NEW\\file2.txt'
```

Виключення `FileNotFoundError` є підкласом класу `OSError` і виконується, якщо зазначений файл не знайдений.

Функція `copyfile()` як результат повертає шлях файлу, куди були скопійовані дані.

Функція copy()

`copy(<Копійований файл>, <Куди копіюємо>)`

Функція дозволяє скопіювати файл разом з правами доступу.

Якщо файл існує, то він буде перезаписаний.

Якщо файл не вдалося скопіювати, виконується виключення `OSError` або одне з виключень, що є підкласом цього класу.

Приклад 23.

```
#Копіювання з правами доступу
import shutil # Підключаємо модуль
my_path = shutil.copy(r"file.txt", r"file3.txt")
print(my_path) #Шлях копіювання
```

Функція `copy()` як результат повертає шлях скопійованого файлу.

Функція copy2()

`copy2(<Копіюємий файл>, <Куди копіюємо>)`

Функція дозволяє скопіювати файл разом з метаданими.

Якщо файл існує, то він буде перезаписаний.

Якщо файл не вдалося скопіювати, виконується виключення `OSError` або одне з виключень, що є підкласом цього класу.

Приклад 24.

```
#Копіювання з метаданими
import shutil # Підключаємо модуль
my_path = shutil.copy2(r"file.txt", r"file4.txt")
print(my_path) #Шлях копіювання
```

Функція `copy2()` як результат повертає шлях скопійованого файлу.

Функція move()

`move(<Шлях до файлу>, <Куди переміщаємо>)`

Функція переміщає файл у зазначене місце з видаленням вхідного (початкового) файлу.

Якщо файл існує, то він буде перезаписаний.

Якщо файл не вдалося перемістити, виконується виключення `OSError` або одне з виключень, що є підкласом цього класу.

Приклад переміщення файлу `file4.txt` у каталог `C:\PYTHON\folder1`:

Приклад 25.

```
#Переміщення
import shutil # Підключаємо модуль
my_path = shutil.copy2(r"file.txt", r"file4.txt")
print(my_path) #Шлях переміщення
```

Функція `move()` як результату повертає шлях переміщеного файлу.

Функція `rename()`

`rename(<Старе ім'я>, <Нове ім'я>)`

Призначена для перейменування файлів. Знаходиться в модулі `os`:

Функція перейменовує файл. Якщо файл не вдалося перейменувати, виконується виключення `Oserror` або одне з виключень, що є підкласом цього класу.

Приклад 26.

```
import os # Подключаем модуль
try:
   os.rename(r"file3.txt", "file4.txt")
except OSError:
   print("Файл не вдалося перейменувати")
else:
   print("Файл успішно перейменований")
```

Результат виконання: Файл успішно перейменований

Функції `remove()` і `unlink()`

remove (<Шлях до файлу>) unlink (<Шлях до файлу>)

Функції дозволяють вилучити файл. Якщо файл не вдалося вилучити, виконується виключення `OSError` або одне з виключень, що є підкласом цього класу.

Приклад 27.

```
import os # Подключаем модуль
#Створення та видалення файлів
f = open(r"file2.txt", "w")
print(f.write("Рядок1\nРядок2"))
f.close()
f = open(r"file4.txt", "w")
print(f.write("Рядок1\nРядок2"))
f.close()
os.remove(r"file2.txt") #Видалення файла
os.unlink(r"file4.txt")
```

Права доступу до файлів і каталогів

В операційних системах сімейства UNIX кожному об'єкту (файлу або каталогу) призначаються права доступу, надавані тим або іншим різновидам користувачів: власнику, групі й решті інших.

Можуть бути призначені наступні права доступу:

-читання.

Користувач може читати файл або каталог

-запис;

Користувач може записувати у файл або каталог

-виконання.

Користувач може запускати виконання файлу

```
>>> os.unlink(r"file4.txt")
```

Позначення прав доступу

`r` – файл можна читати, а вміст каталогу можна переглядати;

`w` – файл можна модифікувати, видаляти й перейменовувати, а в каталозі можна створювати або видаляти файли. Каталог можна перейменувати або вилучити;

`x` – файл можна виконувати, а в каталозі можна виконувати операції над файлами, у тому числі робити в ньому пошук файлів.

Права доступу до файлу визначають записом типу:

```
-rw-r--r--
```

Перший символ (-) означає, що це файл, і не задає прав доступу.

Далі три символи (rw-) задають права доступу для власника: rw означають читання й запис, символ – означає, що права на виконання немає.

Наступні три символи задають права доступу для групи (r--) – тільки читання.

Останні три символи (r--) задають права для всіх інших користувачів – також тільки читання.

Права доступу до каталогу визначають таким рядком:

```
drwxr-xr-x
```

Перша буква (`d`) означає, що це каталог.

Власник може виконувати в каталозі будь-які дії (`rwx`).

Група може тільки читати й виконувати пошук (`r-x`).

Інші користувачі також можуть тільки читати й виконувати пошук (`r-x`).

Для того, щоб каталог можна було переглядати, повинні бути встановлені права на виконання (`x`).

Права доступу можуть позначатися й числом.

Такі числа називаються *маскою прав доступу.*

Число містить три цифри.

Кожна цифра може змінюватися від 0 до 7 у двійковій системі числення.

Перша цифра задає права для власника,

друга – для группы,

третя – для всіх інших користувачів.

Наприклад, права доступу `-rw-r-r--` відповідають числу `644`.

rw позначаємо як 6, тобто 110 у двійковій системі

r позначаємо як 4, тобто 010 у двійковій системі

Зіставимо числа, що входять у маску прав доступу, з двійковим і буквеним записами в таблиці.

Таблиця. Права доступу в різних записах

Вісімкова цифра	Двійковий запис	Буквений запис	Вісімкова цифра	Двійковий запис	Буквений запис
0	000	---	4	100	r--
1	001	--x	5	101	r-x
2	010	-w-	6	110	r-w
3	011	-wx	7	111	rwx

Тепер зрозуміло, що, згідно з даними цієї таблиці, права доступу

rw-r--r--, 110 100 100, 644

мають еквівалентні представлення

Таким чином, якщо право надане, то у відповідній позиції має бути 1, а якщо ні – то 0.

Визначення доступу до файлу

Для визначення доступу до файлу або каталогу призначена функція **access()** з модуля **os**.

Функція має наступний формат:

access(<Шлях>, <Режим>)

Функція повертає True, якщо перевірка пройшла успішно,

Повертає False якщо доступ відсутній.

У параметрі <Режим> можуть бути зазначені наступні константи, що визначають тип перевірки:

os.F_OK – перевірка наявності шляху або файлу:

os.R_OK – перевірка на можливість читання файлу або каталогу;

os.W_OK – перевірка на можливість запису у файл або каталог;

os.X_OK – визначення, чи є файл або каталог виконуваним.

Приклад 28.

```
import os # Підключаємо модуль os
# Перевірка існування файлу
print("file.txt",os.access(r"file.txt", os.F_OK))

# Перевірка існування каталогу
print("C:\PYTHON",os.access(r"C:\PYTHON", os.F_OK))

print("C:\PYTHON_NEW",os.access(r"C:\PYTHON_NEW", os.F_OK))
# каталог не існує
```

Результат виконання:

```
file.txt True
C:\PYTHON True
C:\PYTHON_NEW False
```

Зміна прав доступу

Щоб змінити права доступу з програми, необхідно скористатися функцією **chmod()** з модуля **os**.

Функція має наступний формат:

chmod(<Шлях>, <Права доступу>)

Права доступу задають у вигляді числа, перед яким слід указати комбінацію символів 0o (це відповідає вісімковому запису числа):

```
>>> os.chmod(r"file.txt", 0o777)
# Повний доступ до файлу
```

Замість числа можна вказати комбінацію констант із модуля `stat`. За додатковою інформацією звертайтеся до документації з модуля.