Лекція 26
Робота з файлами (продовження)

Функції модуля os.path

Модуль `os.path` містить додаткові функції, що дозволяють перевірити наявність файлу, одержати розмір файлу й ін.

```
exists (<Шлях>)
exists (<Номер файлу>)      #f.fileno
```

Функція перевіряє зазначений шлях на існування.
Функція повертає `True`, якщо шлях до файлу існує
Функція повертає `False` – якщо шлях до файлу не існує

Параметр `<Шлях>` - це послідовність типу `str`, який використовує зворотні слеші для, як роздільники, для запису послідовності дерева папок.
Параметр `<Номер файлу>` - це число, яке повертає метод fileno().

Перевірка існування файлу за шляхом

Приклад 1.

```
import os.path

print("file.txt",os.path.exists(r"file.txt"))

print("file2.txt",os.path.exists(r"file2.txt"))

print("C:\PYTHON",os.path.exists(r"C:\PYTHON"))

print("C:\P_NEW",os.path.exists(r"C:\P_NEW"))
Результат виконання:
file.txt True
file2.txt False
C:\PYTHON True
C:\P_NEW False
```

Перевірка існування файлу за номером

Приклад 2.

```
import os.path
f=open("file.txt")
a=f.fileno()
p= os.path.exists(a)
print("file.txt",p)

f=open(r"C:\PYTHON\file.txt")
b= os.path.exists(f.fileno())
print(r"C:\PYTHON\file.txt", b)
```

Результат виконання:

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

Функція getsize()

Формат функції: getsize (<Шлях до файлу>)

Функція повертає розмір файлу в байтах. Якщо файл не існує, виконується виключення OSError:

Приклад 3.

```
>>> import os.path
>>> os.path.getsize(r"file.txt")#Файл існує
6
>>> os. path. getsize( r" file2.txt")

FileNotFoundError: [WinError 2] Не вдається
знайти зазначений файл: ' file2.txt
```

Функція getatime () (access time)

Формат функції:getatime (<Шлях до файлу>)

Початок епохи (00:00:00 UTC) 1 січня 1970 року

Функція служить для визначення часу останнього доступу до файлу. Як значення функція повертає кількість секунд, що пройшли з початку епохи. Якщо файл не існує, виконується виключення OSError.

Приклад 4.

```
>>> import os.path
>>> import time
>>> t = os.path.getatime(r"file.txt")
>>> t
1470744657.7352295
>>> time.strftime("%d.%m.%Y %H:%M:%S",time.localtime(t))
'09.08.2016 15:10:57'
```

Функція getctime () (creation time)

Формат функції:getctime (<Шлях до файлу>)

Функція дозволяє довідатися дату створення файлу. Як значення функція повертає кількість секунд, що пройшли з початку епохи.

Якщо файл не існує, виконується виключення OSError.

Приклад 5.

```
>>> import os.path
>>> import time
>>> t = os.path.getctime(r"file.txt")
>>> t
1470744657.7352295
>>> time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(t))
'09.08.2016 15:10:57'
```

Функція getmtime () (modification time)

Формат функції: getmtime (<Шлях до файлу>)

Повертає час останньої зміни файлу. Як значення функція повертає кількість секунд, що пройшли з початку епохи. Якщо файл не існує, виконується виключення `OsError`.

Приклад 6.

```
>>> import os.path
>>> import time
>>> t = os.path.getmtime(r"file.txt")
>>> t
1304044731.265625
>>> time.strftime("%d.% m.% Y %H:%M:%S", time.localtime(t))
'09.08.2016 15:10:57'
```

Функція stat() з модуля os

Формат функції: stat(<Шлях до файлу>)

Функція дозволяє одержати параметри файлу в об'єкті stat_result.

Об'єкт містить десять атрибутів:

st_mode	Тип файлу та режим доступу
st_ino	Номер файлу в структурі Айнод (linux),
st_dev,	Ідентифікатор приладу зберігання файлу
st_nlink,	Визначає місце файлу на диску
st_uid,	Ідентифікатор користувача
st_gid,	Ідентифікатор групи користувача
st_size,	Розмір файлу
st_atime,	Час останнього доступу
st_mtime	Час останньої модифікації
st_ctime.	Час створення

Приклад використання функції `stat ()` наведений у прикладі.

Приклад 7.

```
import os,time
s = os.stat(r"file.txt")
print("{0:20}     {1:7}".format("Атрибути файлу:", "file.txt" ))
print(s)

print("-------------------------------------------------")
print("{0:<30} {1:<20}".format("Розмір файлу file.txt:",s.st_size ))
tc=time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(s.st_ctime))
print("{0:<30} {1:<20}".format("Час створення file.txt:",tc ))
ta=time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(s.st_atime))
print("{0:<30} {1:<20}".format("Час доступу file.txt:",ta ))
tm=time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(s.st_mtime))
print("{0:<30} {1:<20}".format("Час модифікації file.txt:",tm ))
print("-------------------------------------------------")
```

Результати виконання:

Атрибути файлу: file.txt

os.stat_result(st_mode=33206, st_ino=1407374883568982 9, st_dev=1927500709, st_nlink=1, st_uid=0, st_gid=0, st_size=8, st_atime=1480154571, st_mtime=1480159178, st_ctime=1480154571)

Розмір файлу file.txt: 8

Час створення file.txt: 26.11.2016 12:02:51

Час доступу file.txt: 26.11.2016 12:02:51

Час модифікації file.txt: 26.11.2016 13:19:38

Функція `utime()` з модуля os

Оновити час останнього доступу й час зміни файлу дозволяє функція **`utime()`** з модуля os.

utime(<Шлях до файлу>, None)

utime(<Шлях до файлу>,(<Час доступу>, <Час модифікації>))

Параметр `<Шлях до файлу>` це послідовність типу `str`, який використовує зворотні слеші для, як роздільники, для запису послідовності дерева папок.

Параметр 2 = `None`: час доступу й зміни файлу буде поточним.

Параметр 2 - кортеж з нових значень у вигляді кількості секунд, що пройшли з початку епохи.

Якщо файл не існує, виконується виключення `OSError`.

Приклад використання функції `utime()` наведений нижче.

Приклад 8.

```
import os,time
s=os.stat(r"file.txt")
print("{0:20}     {1:7}".format("Атрибути файлу:", "file.txt" ))
print(s)
print("-------------------------------------------------")
ta=time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(s.st_atime))
print("{0:<30} {1:<20}".format("Час доступу file.txt:",ta ))
tm=time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(s.st_mtime))
print("{0:<30} {1:<20}".format("Час модифікації file.txt:",tm ))
print("-------------------------------------------------")
os.utime(r"file.txt", None)
s=os.stat(r"file.txt")
print("{0:20}     {1:7}".format("Встановлення поточного часу ", "file.txt" ))
print(s)
```

```
print("-------------------------------------------------")
ta=time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(s.st_atime))
print("{0:<30} {1:<20}".format("Час доступу file.txt:",ta ))
tm=time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(s.st_mtime))
print("{0:<30} {1:<20}".format("Час модифікації file.txt:",tm ))
print("-------------------------------------------------")

t = time.time() - 600
os.utime(r"file.txt", (t, t))
s=os.stat(r"file.txt")
print("{0:20}     {1:7}".format("Встановити -600сек:", "file.txt" ))
print(s)
print("-------------------------------------------------")
ta=time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(s.st_atime))
print("{0:<30} {1:<20}".format("Час доступу file.txt:",ta ))
tm=time.strftime("%d.%m.%Y %H:%M:%S", time.localtime(s.st_mtime))
print("{0:<30} {1:<20}".format("Час модифікації file.txt:",tm ))
print("-------------------------------------------------")
```

Перетворення шляху до файлу або каталогу

Перетворити шлях до файлу або каталогу дозволяють наступні функції з модуля `os.path`:

`Формат функції: abspath (<Відносний шлях>)`

Функція перетворить відносний шлях в абсолютний, враховуючи місце розташування поточного робочого каталогу.

Приклад 9.

```
import os.path
print("{0:<30} {1:<20}".format("Відносний шлях","file.txt"))
p1 = os.path.abspath(r"file.txt")
print("{0:<30} {1:<20}".format("Абсолютний шлях",p1))
print("-----------------------------------------------")
```

```
print("{0:<30} {1:<20}".format("Відносний шлях в
підкаталог","folder1\\file.txt"))
p2 = os.path.abspath(r"folder1/file.txt")
print("{0:<30} {1:<20}".format("Абсолютний шлях в підкаталог",p2))
print("-----------------------------------------------")

print("{0:<30} {1:<20}".format("Відносний шлях в
надкаталог","..\\file.txt"))
p3 = os.path.abspath(r" .. /file.txt")
print("{0:<30} {1:<20}".format("Абсолютний шлях в надкаталог",p3))
print("-----------------------------------------------")
```

Атрибут sep з модуля os.path

Можна вказувати як прямі, так і зворотні слеші. 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 10.

```
>>> import os.path
>>> os.path.sep
'\\'

>>> import os
>>> os.sep
'\\'
```

Форматування зворотних слешів

При вказівці шляху в Windows слід враховувати, що слеш є спеціальним символом.

1.Тому слеш необхідно подвоювати (екранувати)

2.Замість звичайних рядків використовувати неформатовані рядки з “r”

Приклад 11.

```
>>> "C:\\PYTHON\\file.txt"  # Правильно
'C:\\PYTHON\\file.txt'

>>> r"C:\PYTHON\file.txt"  # Правильно
'C:\\PYTHON\\file.txt'

>>> "C:\PYTHON\file.txt"  # Неправильно! ! !
'C:\\PYTHON\x0cile.txt'
```

3. Можна користуватися прямими слешами з перетворенням через **os.path.abspath**

```
>>> os.path.abspath("C:/LECTURE26/file.txt")
'C:\\LECTURE26\\file.txt'
```

Якщо необхідно мати слеш у кінці рядка, то використовуємо звичайні рядки:

Приклад 12.

```
>>> "C:\\temp\\new\\" # Правильно
'C:\\temp\\new\\'

# Можна слеш видалити
>>> r"C:\temp\new\\"[:-1]
'C:\\temp\\new\\'
```

Функція `isabs()`

Формат функції: `isabs (<Шлях>)`

Функція повертає `True`, якщо шлях є абсолютним,
Функція повертає `False`, якщо шлях не абсолютний.

Приклад 13.

```
>>> os.path.isabs(r"C:\PYTHON\file.txt")
True

>>> os.path.isabs("file.txt")
False
```

Функція `basename()`

Формат функції: `basename (<Шлях>)`

Функція повертає ім'я файлу без шляху до нього:

Приклад 14.

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

>>>os.path.basename(r"C:\PYTHON\folder1")
'folder1'

>>>os.path.basename("C:\\PYTHON\\folder1\\")
''
```

Функція dirname()

Формат функції: dirname (<Шлях>)

Функція повертає шлях до папки, де зберігається файл:

Приклад 15.

```
>>>os.path.dirname(r"C:\PYTHON\folder1\file4.txt")
'C:\\PYTHON\\folder1'

>>> os.path.dirname(r"C:\PYTHON\folder")
'C:\\PYTHON'

>>> os.path.dirname("C:\\PTHON\\folder\\")
'C:\\PTHON\\folder'
```

Функція split (<Шлях>)

Формат функції: split (<Шлях>)

Функція повертає кортеж із двох елементів: шляху до папки, де зберігається файл, і назви файлу:

Приклад 16.

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

>>> os.path.split(r"C:\PYTHON\folder1")
('C:\\PYTHON', 'folder1')

>>> os.path.split("C:\\PYTHON\\folder1\\")
('C:\\PYTHON\\folder1', '')
```

Функція splitdrive()

Формат функції: splitdrive (<Шлях>)

Функція повертає кортеж із двох елементів: ім’я диску та відносний шлях до файлу:

Приклад 17.

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

>>> os.path.splitdrive(r"C:\PYTHON\folder1\folder2 ")
('C:', '\\PYTHON\\folder1\\folder2')
```

Функція `splitext()`

Формат функції: `splitext (<Шлях>)`

Функція повертає кортеж із двох елементів: шляху з назвою файлу, але без розширення, і розширення файлу (фрагмент після останньої крапки):

Приклад 18.

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

>>> os.path.splitext(r"C:\PYTHON\folder1\first.py")
('C:\\PYTHON\\folder1\\first', '.py')
```

Функція `Join()`

Формат функції:

```
Join(<Шлях1>[, ... , <Шляхn>])
```

Функція з'єднує зазначені елементи шляху, за необхідності вставляючи між ними роздільники:

Приклад 19.

```
>>> os.path.join("C:\\", "PYTHON\\folder1", "file.txt")
'C:\\PYTHON\\folder1\\file.txt'

>>> os.path.join(r"C:/", "PYTHON/folder1/", "file.txt")
'C:/PYTHON/folder1/file.txt'.
```

Функція не виправляє тип слеша!!!!

У деяких операційних системах такий тип слеша може бути коректним.

Під Windows такий шлях працювати не буде.

Функція normpath()

Формат функції: normpath(<Шлях>):

Нехай дано шлях:
'C:/PYTHON/folder1/file.txt'.

Щоб цей шлях зробити коректним, необхідно скористатися функцією normpath():

Приклад 20.

```
>>> p = os.path.join(r"C:\\",
"PYTHON/folder1/", "file.txt")

>>> os.path.normpath(p)
'C:\\PYTHON\\folder1\\file.txt'
```

Перенаправлення вводу/виводу

Значення, що повертається методом `fileno()`, завжди буде більшим за число 2:

0 закріплене за стандартним вводом `stdin`,

1 – за стандартним виводом `stdout`,

2 – за стандартним виводом повідомлень про помилки `stderr`.

Усі ці потоки мають деяку подібність із файловими об'єктами.

`stdout` і `stderr` підтримують метод `write()`,

`stdin` – метод `readline()`.

Якщо цим потокам присвоїти посилання на об'єкт, що підтримує файлові методи, то можна перенаправляти стандартні потоки у відповідний файл.

Приклад 21.

```
import sys                    # Підключаємо модуль sys
tmp_out = sys.stdout # Зберігаємо посилання на sys.stdout
f = open(r"file.txt", "w") # Відкриваємо файл на дозапис
sys.stdout = f # Перенаправляємо вивід у файл
print("Пишемо рядок1 у файл")
sys.stdout = tmp_out # Відновлюємо стандартний вивод
print("Пишемо рядок в стандартний вивід")
f.close() # Закрываємо файл
f = open(r"file.txt", "r") # Відкриваємо файл на читання
p = f.read()
print(p)
f.close() # Закриваємо файл
```

Результати виконання:

Пишемо рядок в стандартний вивід

Пишемо рядок1 в файл

Пояснення до попереднього прикладу

1. Зберігаємо у тимчасовій змінній посилання на стандартний вивід `tmp_out`.

```
tmp_out = sys.stdout
```

2. Відкриваємо файл на запис:

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

3. Перенаправляємо вивід файлу на sys.stdout:

```
sys.stdout = f
```

4. Повертаєм вивід в sys.stdout:

```
sys.stdout = tmp_out
```

5. Закриваємо файл

```
f.close()
```

6. Відкриваємо файл file.txt на читання, зчитуємо дані і бачимо, ще записані в нього дані відсутні, оскільки вони були перенаправлені sys.stdout.

Перенаправлення виводу з використанням функції print()

Параметр `file` функції `print`

Функція `print()` прямо підтримує перенаправлення виводу.

Для цього використовується параметр **`file`**, який за замовчуванням посилається на стандартний потік виводу.

Наприклад,записати рядок у файл можна так:

Приклад 22.

```
>>> f = open(r"file.txt", "a")
>>> print("Пишемо рядок4 у файл", file = f)
>>> f.close()
```

Параметр **`flush`** функції `print`

Параметр **`flush`** дозволяє виконати безпосереднє збереження даних із проміжного буфера у файл.

Якщо його значення дорівнює `False` (це значення за замовчуванням), збереження буде виконано лише після закриття файлу або після виклику методу `flush()`.

Якщо **`flush=True`**, то збереження файлу буде після кожного виклику функції `print()`

Приклад 23.

```
import io
f = open(r"file.txt", "a+")
print("Пишемо рядок5 у файл", file = f, flush = True)
print("Пишемо рядок6 у файл", file = f, flush = True)
f.seek(0,io.SEEK_SET)
print(f.read())
f.close()
```

Стандартний ввід `stdin`

`sys.stdin` використовується для стандартного інтерактивного вводу даних:

```
sys.__stdin__
sys.__stdout__
sys.__stderr__ ¶
```

Це об'єкти які містять оригінальні значення для `stdin, stdout` та `stderr` на початку програми.

Стандартний ввід `sys.stdin` та `sys.stderr` також можна перенаправляти.

У цьому випадку функція `input()` буде читати один рядок з файлу при кожному виклику.

При досягненні кінця файлу виконується виключення `EOFError`.

Приклад 24.

```
import sys
tmp_in = sys.stdin # Зберігаємо посилання на sys.stdin
f = open(r"file.txt", "r")# Відкриваємо файл на читання
sys.stdin = f # Перенаправляємо ввід
while True:
    try:
        line = input()# Читаємо рядок з файлу
        print(line) # Виводимо рядок
    except EOFError: # Якщо досягнутий кінець файлу,
        break  # виходимо з циклу
sys.stdin = tmp_in # Відновлюємо стандартний ввід
f. close ()# Закриваємо файл
```

```
Результати виконання:
Пишемо рядок1 в файл
Пишемо рядок2 в файл
Пишемо рядок3 в файл
Пишемо рядок4 в файл
```

Метод isatty()

Якщо необхідно довідатися, чи посилається стандартний ввід на термінал, можна скористатися методом `isatty()`.

Метод повертає `True`, якщо об'єкт посилається на термінал, і `False` – якщо ні.

Приклад 25.

```
import sys,os
if  sys.stdin.isatty():
    print("We use TTY")
f = open(os.path.abspath("C:/LECTURE26/file.txt"), "r")
tmp_in = sys.stdin # Зберігаємо посилання на sys.stdin
sys.stdin = f  # Перенаправляємо ввід
if  not sys.stdin.isatty():
    print("We use file", f.name)
sys.stdin = tmp_in # Відновлюємо стандартний ввід
f.close() # Закриваємо файл
```

Командний рядок для перенаправлення вводу/виводу

Перенаправляти стандартне ввід/вивід можна також за допомогою командного рядка.

Для прикладу створимо в папці `C:\LECTURE26` файл **`redir.py`** з кодом, наведеним у прикладі.

Приклад 26.

```
while True:
    try:
        line = input ()
        print(line)
    except EOFError:break
```

Результат виконання:

```
ЩО ВВОДИМО, те й виводимо
ЩО ВВОДИМО, те й виводимо
```

1. Запускаємо командний рядок, набравши `Пуск->Виконати->cmd->OK`.
2. Переходимо в папку зі скриптом, виконавши команди:

```
cd\
cd LECTURE26
```

3. Тепер виведемо вміст створеного раніше текстового файлу `file.txt` (його вміст може бути будь-яким), виконавши команду:

```
python.exe redir_in.py < file.txt
```

Перенаправляти стандартний вивід у файл можна у аналогічний спосіб. Тільки в цьому випадку символ `<` необхідно замінити символом `>`.

Змінимо файл `redir_out.py` у такий спосіб:

Приклад 27.

```
print ("Вивід перенаправлено у файл")
# Цей рядок буде записаний у файл
```

Тепер перенаправляємо вивід у файл `file.txt`, виконавши команду:

```
python.exe redir_out.py > file.txt
```

У цьому режимі файл `file.txt` буде перезаписаний.

```
Результат виконання:
file.txt: Вивід перенаправлений у файл
```

Якщо необхідно додати результат у кінець файлу, слід використовувати символи **>>**.

Приклад дозапису в файл:

Приклад 28.

Запишемо у файл `redir_out.py:`

```
print ("Дані дописані у файл")
```

Перейдемо знову в командний рядок

```
python.exe redir_out.py >> file.txt
Результат виконання:
file.txt: Вивід перенаправлений у файл
          Дані  дописані у файл
```