Лекція 27

Робота з файлами (завершення)

Збереження об'єкту у файл

Зберегти об'єкти у файл і надалі відновити об'єкти з файлу дозволяють модулі `pickle` і `shelve`.

Функція dump():

dump (<Об'єкт>, <Файл>)

Функція виконує серіалізацію об'єкта й записує дані в зазначений файл.

У параметрі <Файл> вказується файловий об'єкт, відкритий на запис у **бінарному режимі**. Приклад збереження об'єкта у файл:

Приклад 1.

```
import pickle
f = open(r"file.txt", "wb")
obj = ["Рядок", (2, 3)]
pickle.dump(obj, f)
f.close()
```

Функція load()

Функція **load()** читає дані з файлу й перетворює їх в об'єкт. У параметрі <Файл> вказується файловий об'єкт, відкритий на читання в **бінарному режимі**.

Формат функції:

load(<Файл>)

Приклад відновлення об'єкта з файлу:

Приклад 2.

```
import pickle
#Зчитування об'єкта з файла
f = open(r"file.txt", "rb")
obj = pickle.load(f)
print(obj)
f.close()
```

Результат:

```
['Рядок', (2, 3)]
```

Збереження кількох об’єктів у файлі

В один файл можна зберегти відразу кілька об'єктів, послідовно викликаючи функцію **dump()**.

Приклад збереження декількох об'єктів:

Приклад 3.

```
import pickle
#Збереження кількох об'єктів у файлі
obj1 = ["Рядок", (2, 3)]
obj2 = (1, 2)
f = open(r"file.txt", "wb")
# Зберігаємо перший об'єкт
pickle.dump(obj1, f)
# Зберігаємо другий об'єкт
pickle.dump(obj2, f)
f. close()
```

Відновлення кількох об’єктів з файлу

Для відновлення об'єктів необхідно кілька раз викликати функцію `load()`.

Приклад 4.

```
import pickle
f = open(r"file.txt", "rb")
# Відновлюємо перший об'єкт
obj1 = pickle.load(f)
# Відновлюємо другий об'єкт
obj2 = pickle.load(f)
print( obj1, obj2)
f.close()
```

Результат:

```
['Рядок', (2, 3)] (1, 2)
```

Класи Pickler і Unpickler

Зберегти об'єкт у файл можна також за допомогою методу **dump**(<Об'єкт>) класу **Pickler**. Конструктор класу має наступний формат:

```
Pickler(<Файл>)
```

Збереження

Приклад збереження об'єкта у файл:

Приклад 5.

```
import pickle
#Збереження об'єкта методом Pickle.dump
f = open(r"file.txt", "wb")
obj = ["Рядок", (2, 3)]
pkl = pickle.Pickler(f)
pkl.dump(obj)
f.close()
```

Відновлення

Відновити об'єкт із файлу дозволяє метод **load()** із класу **Unpickler**.

Формат конструктора класу:

```
Unpickler(<Файл>)
```

Приклад відновлення об'єкта з файлу:

Приклад 6.

```
import pickle
#Відновлення об'єкта методом load класу Unpickler
f = open(r"file.txt", "rb")
unp = pickle.Unpickler(f)
unp.load()
print(obj)
f.close()
```

Результат:

```
['Рядок', (2, 3)]
```

Перетворення об’єкт у послідовність байтів

Модуль `pickle` дозволяє також перетворити об'єкт у послідовність байтів і відновити об'єкт із послідовності. Для цього призначена функція:

```
dumps(<Об'єкт>[, <Протокол> ])
```

Функція виконує серіалізацію об'єкта й повертає послідовність байтів спеціального формату.

Формат залежить від зазначеного протоколу –числа від 0 до 4 у порядку від більш старих до більш нових і досконалих.

За замовчуванням використовується протокол 4.

Приклад перетворення списку й кортежу:

Приклад 7.

```
import pickle
obj1 = [1, 2, 3, 4, 5]   # Список
obj2 = (6, 7, 8, 9, 10) # Кортеж
bin_obj1 = pickle.dumps(obj1)
print(bin_obj1)
bin_obj2 = pickle.dumps(obj2)
print(bin_obj2)
```

Результат:

```
b'\x80\x03]q\x00(K\x01K\x02K\x03K\x04K\x05e.'
b'\x80\x03(K\x06K\x07K\x08K\tK\ntq\x00.'
```

Відновлення послідовності байтів у об’єкт

Функція перетворює послідовність байтів спеціального формату в об'єкт.

```
loads(<Послідовність байтів>)
```

Приклад відновлення списку й кортежу:

Приклад 8.

```
import pickle
obj1 = pickle.loads(b'\x80\x03]q\x00(K\x01K\x02K\x03K\x04K\x05e.')
print(obj1)
obj2 = pickle.loads(b'\x80\x03(K\x06K\x07K\x08K\tK\ntq\x00.')
print(obj2)
```

Результат:

```
[1, 2, 3, 4, 5]
(6, 7, 8, 9, 10)
```

Модуль shelve

Модуль `shelve` дозволяє зберігати об'єкти під певним ключем.

Ключ задають у вигляді рядка.

Доступ до об’єктів подібний до доступу у словниках.

Відкриття файла з набором об'єктів виконує функція

`shelve.open().`

Функція має наступний формат:

```
shelve.open(<Шлях до файлу>[, flag="c"])
```

Параметр `flag` задає режими відкриття файлу:

`r` – тільки читання;

`w` – читання й запис;

`c` – читання й запис (значення за замовчуванням). Якщо файл не існує, він буде створений;

`n` – читання й запис. Якщо файл не існує, він буде створений. Якщо файл існує, він буде перезаписаний.

Методи об’єкту `DbfilenameShelf`

Функція **`open()`** повертає об'єкт

`shelve``.DbfilenameShelf`

За допомогою даного об’єкту, проводиться подальша робота з базою даних.

Методи:

`close()` – закриває файл із базою даних.

Для прикладу створимо файл і збережемо в ньому список і кортеж.

Приклад 9.

```
import shelve #Підключаємо модуль

db = shelve.open("db1")#Відкриваємо файл

#Зберігаємо список
db["obj1"] = [1, 2, 3, 4, 5]

#Зберігаємо кортеж
db["obj2"] = (6, 7, 8, 9, 10)

# Вивід значень
print(db["obj1"])
print(db["obj2"])

#Закриваємо файл
db.close()
```

Основні методи `shelve.DbfilenameShelf`

`keys()` – повертає об'єкт із ключами;

`values()` – повертає об'єкт зі значеннями;

`items()` – повертає об'єкт-ітератор, який на кожній ітерації генерує кортеж, що містить ключ і значення.

Приклад 10.

```
import shelve #Підключаємо модуль
db = shelve.open("MyDatabase")
print(db.keys())
print(db.values())
print(db.items())
print("-----------------------")
a = list(db.keys()), list(db.values())
print(a)
b = list(db.items())
print(b)
print("-----------------------")
db.close()
```

Результат роботи:

KeysView(<shelve.DbfilenameShelf object at 0x005D2050>)

ValuesView(<shelve.DbfilenameShelf object at 0x005D2050>)

ItemsView(<shelve.DbfilenameShelf object at 0x005D2050>)

(['obj2', 'obj1'], [(6, 7, 8, 9, 10), [1, 2, 3, 4, 5]])

[('obj2', (6, 7, 8, 9, 10)), ('obj1', [1, 2, 3, 4, 5])]

Метод get ()

```
get(<Ключ>[, <Значення за замовчуванням> ])
```

Якщо ключ присутній, то метод повертає значення, відповідне до цього ключа.
Якщо ключ відсутній, то повертається значення None або значення, зазначене в другому параметрі;

Приклад 11.

```
import shelve      #Підключаємо модуль
db = shelve.open("MyDatabase")#Відкриваємо файл
db["obj1"] = [1, 2, 3, 4, 5]
db["obj2"] = (6, 7, 8, 9, 10)
db["obj3"] = "Рядок"
a=db.get("obj3")
print(a)
db.close()
Результат: Рядок
```

Метод setdefault

setdefault(<Ключ>[,<Значення за замовчуванням>])

1.Якщо ключ присутній, то метод повертає значення, відповідне до цього ключа.

2.Якщо ключ відсутній, створюється новий елемент зі значенням, зазначеним у другому параметрі, і як результат повертається це значення.

3.Якщо другий параметр не зазначений, значенням нового елемента буде None.

Приклад 12.

```
import shelve #Підключаємо модуль
db = shelve.open("MyDatabase")#Відкриваємо файл
db["obj1"] = (9, 14)
d={"c":1,"k":2}
a=db.setdefault("obj1")
b=db.setdefault("obj4",d)
db.close()
print(a, b)
(9, 14) {'k': 2, 'c': 1}
```

Метод pop()

pop(<Ключ> [, <Значення за замовчуванням>])

1.Видаляє елемент із зазначеним ключем і повертає його значення.

2.Якщо ключ відсутній, повертається значення із другого параметра.

3.Якщо ключ відсутній, і другий параметр не зазначений, то виконується виключення Keyerror.

Приклад 13.

```
import shelve #Підключаємо модуль
db = shelve.open("MyDatabase")#Відкриваємо файл
db["obj1"] = (9, 14)
d={"c":1,"k":2}
a=db.pop("obj1")
b=db.pop("obj4",d)
db.close()
print(a, b)
Результат: (9, 14) {'k': 2, 'c': 1}
```

Приклад 14.

```
import shelve #Підключаємо модуль
db = shelve.open("MyDatabase")#Відкр. Файл
db["obj1"] = (9, 14)
db["obj2"] = (15, 17)
db["obj3"] = "string"
d={"c":1,"k":2}
a=db.pop("obj1")
b=db.pop("obj4",d)
m=list(db.items())
db.close()
print(a, b)
print(m)
Результат:
(9, 14) {'k': 2, 'c': 1}
[('obj3', 'string'), ('obj2', (15, 17))]
```

Метод `popitem()`

`popitem()` – видаляє довільний елемент і повертає кортеж з ключа й значення. Якщо файл порожній, виконується виключення `Keyerror`.

Приклад 15.

```
import shelve
db = shelve.open("db1")
db["obj1"] = (9, 14)
db["obj2"] = (15, 17)
db["obj3"] = "string"
d={"c":1,"k":2}
a=db.popitem(); b=db.popitem()
m=dict(db.items())
db.close()
print(a, b);print(m)
Результат:
('obj1', (9, 14)) ('obj2', (15, 17))
{'obj3': 'string'}
```

Метод `clear()`

`clear()` – видаляє всі елементи.

Метод нічого не повертає як значення.

Приклад 16.

```
import shelve
db = shelve.open("MyDatabase")
db["obj1"] = (9, 14)
db["obj2"] = (15, 17)
db["obj3"] = "string"
d={"c":1,"k":2}
db.clear()
m=dict(db.items())
db.close()
print(m)
Результат:
{}
```

Метод `update()`

`update()` – додає елементи. Метод змінює поточний об'єкт і нічого не повертає. Якщо елемент із зазначеним ключем уже присутній, то його значення буде перезаписано.

Формати методу:

`update`(<Ключ1>=<Значення1>[, ... , <Ключn>=<Значенняn>])

`update`(<Словник>)

`update`(<Список кортежів із двома елементами>)

`update`(<Список списків із двома елементами>)

Працюють також:

Функція **`len()`** - для одержання кількості елементів.

Оператор **`del`** для видалення певного елемента,

Оператори **`in`** і **`not in`** для перевірки існування або неіснування ключа.

Приклад 17.

```
import shelve
#Застосування методу update() у різних варіантах
db = shelve.open("MyDatabase")
db.update(obj1=(9, 14),obj2=(15, 17))
db.update({"obj3":4})
db.update([("obj4",4),("obj5",5)])
db.update([["obj7",7],["obj6",6]])
m=dict(db.items())
db.close()
print(m)
Результат:
{'obj7': 7, 'obj2': (15, 17), 'obj3': 4,
'obj5': 5, 'obj6': 6, 'obj1': (9, 14),
'obj4': 4}
```

Приклад 18.

```
import shelve
#Перевірка наявності та видалення
db = shelve.open("MyDatabase")
print("{0:<20} {1:<20}".format("Розмір бази:",len(db)))
a="obj1" in db
print("{0:<20} {1:<20}".format("Наявність об'єкта obj1:",a))
del db["obj1"] # Видалення елемента
a="obj1" in db
print("{0:<20} {1:<20}".format("Наявність об'єкта obj1:",a))
b="obj1" not in db
print("{0:<20} {1:<20}".format("Відсутність об'єкта:",b))
db.close()
```

```
Розмір бази:          7
Наявність об'єкта obj1: 1
Наявність об'єкта obj1: 0
Відсутність об'єкта:    1
```

Функції для роботи з каталогами з модуля os

`getcwd()` –повертає поточний робочий каталог.

(get current work directory)

Від значення, що повертається цією функцією, залежить перетворення відносного шляху в абсолютний.

Крім того, важливо пам'ятати, що поточним робочим каталогом буде каталог, з якого запускається файл, а не каталог з файлом, що виконується.

Приклад 19.

```
import os
wd=os.getcwd()# Поточний робочий каталог
print(wd)
```

Результат:

```
C:\LECTURE27
```

`chdir (<Ім'я каталогу>)` – робить зазначений каталог поточним:

Приклад 20.

```
import os
# chdir() робить зазначений каталог поточним
print("Змінили робочий каталог")
os.chdir("C:\\LECTURE27\\folder1\\")
print(os.getcwd()) # Поточний робочий каталог
print("Повернули робочий каталог")
os.chdir("C:\\LECTURE27\\")
print(os.getcwd()) # Поточний робочий каталог
```

Результат:

```
Змінили робочий каталог
C:\LECTURE27\folder1
Повернули робочий каталог
C:\LECTURE27
```

`mkdir(<Ім'я каталогу>[, <Права доступу>])` – створює новий каталог із правами доступу, зазначеними в другому параметрі.

Права доступу задають вісімковим числом (значення за замовчуванням `0o777`).

Приклад створення нового каталогу в поточному робочому каталозі:

Приклад 21.

```
import os
try:
    os.mkdir("newfolder")#Створення каталогу
    os.mkdir("newfolder\\subfolder")
    print("Каталог створено")
except FileExistsError:
    print("Каталог уже існує")
```

`rmdir (<Ім'я каталогу>)` – видаляє порожній каталог. Якщо в каталозі є файли або зазначений каталог не існує, виконується виключення – підклас класу `OSError`.

Вилучимо каталог `newfolder`:

Приклад 22.

```
import os
# rmdir видаляє пусті існуючі каталоги
try:
    os.rmdir("newfolder\\subfolder") # Видалення
print("Видалили пустий каталог newfolder\\subfolder")
    os.rmdir("newfolder") # Видалення каталогу
    print("Видалили пустий каталог newfolder")
except OSError:
    print("Каталог не існує або він не пустий  ")
```

`listdir(<Шлях>)` – повертає список об'єктів у зазначеному каталозі:

Приклад 23.

```
import os

# listdir створює список об'єктів, розміщених у каталозі
dl = os.listdir("C:\\LECTURE27\\folder1\\")
print(dl)
```

Результат:

```
['file.txt', 'file2.txt', 'file3.txt',
'file4.txt']
```

`walk()` – дозволяє обійти дерево каталогів.

Формат функції:

```
walk(<Початковий каталог>[, topdown=True])
```

`walk()` повертає об'єкт

На кожній ітерації через цей об'єкт доступний кортеж із трьох елементів:

1 елемент-поточний каталог,

2 елемент-список каталогів,

3 елемент список файлів, що містяться у каталозі.

Параметр `topdown` задає послідовність обходу каталогу.

`Topdown=True` - значення за замовчуванням.

У цьому випадку пошук зверху вниз

Приклад 24.

```
import os
for (p, d, f) in os.walk("C:\\LECTURE27\\"):
    print("Поточний каталог")
    print(p)
    print("Список каталогів")
    print(d)
    print("Список файлів у каталозі")
    print(f,"\n")
```

Результати:

```
Список каталогів
['folder1', '__pycache__']
Список файлів у каталозі
['Ex1.py', 'Ex10.py', …]
```

```
Поточний каталог
C:\LECTURE27\folder1
Список каталогів
['folder2']
Список файлів у каталозі
['file3.txt', 'file4.txt']

Поточний каталог
C:\LECTURE27\folder1\folder2
Список каталогів
['folder3']
Список файлів у каталозі
['file.txt']

Поточний каталог
C:\LECTURE27\folder1\folder2\folder3
Список каталогів
[]
Список файлів у каталозі
['file2.txt']
```

`Topdown = False` послідовність обходу знизу вверх:

Приклад 25.

```
import os
for (p, d, f) in os.walk("C:\\LECTURE27\\"):
    print("Поточний каталог")
    print(p)
    print("Список каталогів")
    print(d)
    print("Список файлів у каталозі")
    print(f,"\n")
```

Результати:

```
Поточний каталог
C:\LECTURE27\folder1\folder2\folder3
Список каталогів
[]
Список файлів у каталозі
['file2.txt']
```

```
Поточний каталог
C:\LECTURE27\folder1\folder2
Список каталогів
['folder3']
Список файлів у каталозі
['file.txt']

Поточний каталог
C:\LECTURE27\folder1
Список каталогів
['folder2']
```

Завдяки такій послідовності обходу каталогів можна вилучити всі вкладені файли й каталоги. Це особливо важливо при видаленні каталогу, тому що функція `rmdir()` дозволяє вилучити тільки порожній каталог.

Приклад очищення дерева каталогів:

Приклад 26.

```
import os
for (p, d, f) in
os.walk("C:\\LECTURE27\\folder1\\",  False):
    for file_name in f:#Видаляємо всі файли
        print(file_name)
        print(os.path.join(p, file_name))
        os.remove(os.path.join(p, file_name))

   for dir_name in d:#Видаляємо всі каталоги
        os.rmdir(os.path.join(p, dir_name))
```

*УВАГА!Дуже обережно використовуйте цей код. Якщо як перший параметр у функції `walk()` указати кореневий каталог диска, то **всі** наявні в ньому файли й каталоги **будуть вилучені**.*

Функція `rmtree()` з `shutil`

Вилучити дерево каталогів дозволяє також функція `rmtree()` з модуля `shutil`.

Функція має наступний формат:

`rmtree``( <Шлях> [, <Обробка помилок> [, <Оброблювач помилок>]] )`

`<Обробка помилок>` зазначаємо `True`, то помилки будуть зігноровані.

`<Обробка помилок>` зазначаємо `False` (значення за замовчуванням), то в третьому параметрі можна вказати посилання на функцію-оброблювач.

Ця функція буде викликатися при виникненні виключення:

Приклад 27.

```
import shutil
try:
 s = "C:\\LECTURE27\\folder1\\"
 shutil.rmtree(s, False)
except FileNotFoundError:
    print("Каталог ",s,"не знайдено")
```

Рузультат:

```
Каталог  C:\LECTURE27\folder1\ не знайдено
```

```
import shutil
s = "C:\\LECTURE27\\folder1\\"
shutil.rmtree(s, True)
```

Рузультат:

```
Не буде ніякого виводу
```

`normcase (<каталог>)` – перетворює заданий шлях до каталогу до вигляду, придатному для використання в поточній операційній системі. В Windows перетворює усі прямі слеші у зворотні. Також у всіх системах приводить усі букви шляху до нижнього регістру.

Приклад 28.

```
from os.path import normcase
pt = normcase(r"c:/LECTURE27/file.txt")
print(pt)
```

Рузультат:

```
c:\lecture27\file.txt
```

`isdir(<Об'єкт>)` – повертає `True`, якщо об'єкт є каталогом, і `False` – якщо ні:

Приклад 29.

```
import os.path
if not os.path.isdir(r"C:\LECTURE27\file.txt"):
    print(r"C:\LECTURE27\file.txt- це файл")
if os.path.isdir("C:\\LECTURE27\\"):
    print(r"C:\LECTURE27\- це каталог")
```

Рузультат:

```
C:\LECTURE27\file.txt- це файл
C:\LECTURE27\- це каталог
```

`isfile(<Объект>)` – повертає `True`, якщо об'єкт є файлом, і `False` – якщо ні:

Приклад 30.

```
import os.path
if  os.path.isfile(r"C:\LECTURE27\file.txt"):
   print(r"C:\LECTURE27\file.txt- це файл")
if not os.path.isfile("C:\\LECTURE27\\"):
   print(r"C:\LECTURE27\- це каталог")
```

Рузультат:

```
C:\LECTURE27\file.txt- це файл
C:\LECTURE27\- це каталог
```

`glob (<Шлях>)` використовується для обмеження списку певними критеріями. Функція дозволяє вказати в шляху наступні спеціальні символи:

`?` - будь-який одиночний символ;

`*` - будь-яка кількість символів;

`[<Символи>]` – дозволяє вказати символи, які повинні бути на цьому місці в шляху.

Можна перелічити символи.
Можна вказати діапазон через дефіс.

Як значення функція повертає список шляхів до об'єктів, співпадаючих із шаблоном.

Приклад 31.

```
import glob
print("Всі файли з розширенням txt")
pa = glob.glob("C:\\LECTURE27\\folder1\\*.txt")
print(pa)
print("Всі файли з розширенням html")
pb = glob.glob("C:\\LECTURE27\\folder1\\*.html") # Абс. шлях
print(pb)
pc = glob.glob("folder1/*.html") # Відносний шлях
print(pc)
print("Всі файли з цифрою 1 наприкінці назви")
pd = glob.glob("C:\\LECTURE27\\folder1\\*1.*")
print(pd)
```

Результат:

Всі файли з розширенням txt

['C:\\LECTURE27\\folder1\\file0.txt', 'C:\\LECTURE27\\folder1\\file1.txt']

Всі файли з розширенням html

['C:\\LECTURE27\\folder1\\index1.html', 'C:\\LECTURE27\\folder1\\index2.html']

['folder1\\index1.html', 'folder1\\index2.html']

Всі файли з цифрою 1 наприкінці назви

['C:\\LECTURE27\\folder1\\file1.txt', 'C:\\LECTURE27\\folder1\\index1.html']

Спеціальні символи можуть бути зазначені не тільки в назві файлу, але й в іменах каталогів у шляху. Це дозволяє переглядати одразу кілька каталогів у пошуках об'єктів, відповідних до шаблону.

Приклад 32.

```
import glob
p = glob.glob("C:\\LECTURE27\\*\\*.html")
print(p)
```

Результат:

```
['C:\\LECTURE27\\folder1\\index1.html',
'C:\\LECTURE27\\folder1\\index2.html']
```

Виключення, які виникають при файлових операціях

При вивченні виключень зазначалось, що функції й методи, що здійснюють файлові операції, при виникненні позаштатних ситуацій викликають виключення класу `OSError` або одне з виключень, що є його підкласом.

Виключень-підкласів класу `OSError` багато. Ми розглянемо ті з них, що стосуються саме операцій з файлами й папками:

`BlockingIOError` – не вдалося заблокувати об'єкт (файл або потік вводу/виводу);

`ConnectionError` – помилка мережного з'єднання. Може виникнути при відкритті файлу по мережі. Є базовим класом для ряду інших виключень більш високого рівня, описаних у документації по `Python`;

`FileExistsError` – файл або папка із заданим іменем уже існують;

`FileNotFoundError` – файл або папка із заданим іменем не виявлені;

`InterruptedError` – файлова операція несподівано перервана з якої-небудь причини;

`IsADirectoryError` – замість шляху до файлу зазначений шлях до папки;

`NotADirectoryError` – замість шляху до папки зазначений шлях до файлу;

`PermissionError` – відсутні права на доступ до зазначеного файлу або папки;

`TimeoutError` – вийшов час, відведений системою на виконання операції.

Приклад коду, що обробляє деякі із зазначених виключень, наведений нижче.

Приклад 33.

```
try:
    open(r"C:\temp\new\file.txt")
except FileNotFoundError:
    print("Файл відсутній")
except IsADirectoryError:
    print("Це не файл, а папка")
except PermissionError:
    print("Відсутні права на доступ до
файлу")
except OSError:
    print("Невизначена помилка відкриття
файлу")
```