Работа с популярными форматами файлов (json, zip, xml и т.д.)

План урока

1 Манипуляции с файлами и папками

2 Модули os и os.path

3 Модуль shutil (shell utilities)

4 Zip-архивы

5 Формат JSON

Аннотация

В уроке рассказывается об основных средствах манипуляции с файлами и папками в Питоне, а также о работе с файлами специализированных форматов: zip-архивах и JSON.

1. Манипуляции с файлами и папками

Почти все операционные системы поддерживают технологию иерархической (или древовидной) организации файловых систем. Кроме понятия **файл** существует также и понятие **папки** (или каталога). Папка — это контейнер, содержащий файлы, причём папки могут быть вложены друг в друга. Файловая система представляет своеобразное дерево с вершинами-папками и листьями-файлами. В UNIX-подобных операционных системах, благодаря концепции «всё есть файл», папка является также специальным файлом.

Действия с файлами и папками могут быть специфичными для операционной системы, поэтому нужно внимательно изучать документацию к той или иной библиотеке языка. Однако большинство обычных операций с файловой системой для большинства современных операционных систем универсальны.

Помимо работы с содержанием файла, в Python есть средства и способы по работе с файловой системой в целом, а именно:

— управление местоположением файла (копирование и перемещение);

— создание и удаление файла;

— обход файловой системы;

— получение метаданных файлов;

— и т.д.

Мы начнём наш обзор возможностей с модуля **os**, потом рассмотрим модуль **os.path** и модуль **shutil**.

2. Модули os и os.path

Модуль **os** содержит в себе некоторые уникальные, зависящие от конкретной операционной системы функции. Посмотрим, что же в нём находится? Нам поможет функция **dir**.

```
import os
print(dir(os))
```

Рассмотрим некоторые полезные функции из этого модуля, относящиеся к файлам и папкам. Остальные возможности вы можете почерпнуть из документации самостоятельно.

Функция **os.name()** содержит тип операционной системы, в которой выполняется наша программа, в данном случае posix. Допустимые значения:

— posix,

— nt,

— java.

```
print(os.name)

----------------------------------------------------

posix
```

Чтобы узнать имя текущего каталога надо вызвать функцию **os.getcwd()**:

```
print(os.getcwd())

----------------------------------------------------

/Users/some_user/ЯндексЛицей/Уроки/materials/2_year/11
```

А чтобы сменить текущий каталог — функцию **os.chdir()**:

```
os.chdir('files')
print(os.getcwd())

----------------------------------------------------

/Users/some_user/ЯндексЛицей/Уроки/materials/2_year/11/files
```

Проверить, существует ли файл, доступен ли файл для чтения или записи можно с помощью функции **os.acceess()**:

```
os.access("1.txt", os.F_OK)
os.access("1.txt", os.R_OK)
os.access("такого файла нет.txt", os.F_OK)

----------------------------------------------------

    True
    True
    False
```

Флаги **W_OK**, **R_OK**, **F_O**K отвечают соответственно за возможность записи, чтения, а также за факт существования файла.

Очень полезной может быть функция рекурсивного прохода по всем папкам в заданной папке:

Для примера рассмотрим следующую файловую структуру, начиная с каталога files:

```
| 1.txt
├──1
├──2
└──3
          | Описание.txt
          |
          └──files
                └──csvs
                                данные.csv##
```

```
for currentdir, dirs, files in os.walk('files'):
    print(currentdir, dirs, files)

----------------------------------------------------

    files ['1', '2', '3'] ['1.txt', 'Icon\r']
    files/1 [] ['Icon\r']
    files/2 [] ['Icon\r']
    files/3 ['files'] ['Icon\r', 'Описание.txt']
    files/3/files ['csvs'] []
    files/3/files/csvs [] ['Icon\r', 'данные.csv']
```

Мы видим, что функция **os.walk()** возвращает кортеж из трёх элементов.

Рекурсивно перебирая папки, у нас всегда есть полное имя текущего каталога (относительно того, откуда стартовали), список папок данного каталога и список его файлов.

Например, когда оказались в папке 3, то текущий каталог — это files/3, список папок — это ['files'], а список файлов — ['Icon\r', 'Описание.txt'].

Посмотрим теперь, какие функции нам предлагает модуль **os.path**:

```
print(dir(os.path))
```

С помощью функции **os.path.exists()** можно проверить, существует ли файл, а с помощью функций **os.path.isfile()** и **os.path.isdir()** определить, какого он типа: «настоящий» или папка (здесь работает идеология UNIX).

Как видите, одну и ту же задачу можно выполнить по-разному, используя функционал различных библиотек.

```
os.path.exists('files/3')
os.path.isfile('files/3')
os.path.isdir('files/2')

-----------------------------------------------------

True
False
True
```

Функция **os.path.abspath()** возвратит абсолютный путь по относительному:

```
os.path.abspath('1.txt')

-----------------------------------------------------

'/Users/some_user/ЯндексЛицей/Уроки/materials/2_year/11/1.txt'
```

А функция **os.path.dirname()** — полное имя каталога, в котором находится файл.а

```
os.path.dirname('1.txt')
os.path.dirname('3/files/csvs/данные.csv')

-----------------------------------------------------

''
'3/files/csvs'
```

3. Модуль shutil (shell utilities)

Ещё один полезный модуль — **shutil** (правда весёлое название?). В нём содержатся несколько высокоуровневых функций для манипуляции с файлами: копирование, перемещение, удаление и другие. Всё то, что обычно делается через файловые менеджеры ОС, например, через проводник.

Чтобы скопировать файл, надо использовать функцию **shutil.copy()**. В ней необходимо указать два **обязательных** параметра:

— источник (что копируем);

— приёмник (куда копируем).

Оба параметра — строки.

```
import shutil
shutil.copy('files/3/Описание.txt', 'files/Копия.txt')

--------------------------------------------------

'files/Копия.txt'
```

Заметьте, что если файл-приёмник уже существовал, то он будет **перезаписан**. Кроме того, файл-источник и файл-приёмник не могут совпадать, то есть файл **нельзя** скопировать в себя же.

Удаление папки со всем её содержимым выполняется функцией **shutil.rmtree()** (от английского remove tree). В качестве параметра передаётся полный путь до папки.

```
shutil.rmtree('Путь до папки')
```

Перенос папки (со всем ее содержимым) в новое место осуществляется функцией **shutil.move()**, которая по своему виду похожа на shutil.copy(), только после копирования она «заметает свои следы».

```
shutil.move(старое_место, новое_место)
```

Также функции модуля **shutil** дают возможность работать с некоторыми типами зарегистрированных в системе архивов, создавая их или распаковывая:

```
shutil.get_archive_formats()

--------------------------------------------------

    [('bztar', "bzip2'ed tar-file"),
     ('gztar', "gzip'ed tar-file"),
     ('tar', 'uncompressed tar file'),
     ('xztar', "xz'ed tar-file"),
     ('zip', 'ZIP file')]
```

```
shutil.make_archive('archive', 'zip', root_dir='files')

--------------------------------------------------
```

```
'/Users/some_user/ЯндексЛицей/Уроки/materials/2_year/11/archive.zip'
```

В последнем примере мы создали архив с именем archive типа zip, положив него всё содержимое каталога files. Сам архив при этом будет размещён в текущем каталоге.

4. Zip-архивы

Раз уж мы заговорили об архивах, то задержимся на них немного подольше.

Работе с архивами посвящена соответствующая глава документации. В современном мире применяется большое количество архиваторов: zip, 7z, rar и т.д. Мы пока остановимся только на zip-архивах по двум причинам:

— Это самый распространённый и свободный формат архива. (Под термином **свободный** мы понимаем то, что алгоритм опубликован и свободен в том числе от коммерческих отчислений автору);

— Несколько типов файлов, например, docx, pptx, jar являются по сути форматами на основе zip-архивов. Сделано это было потому, что удобнее все ресурсы документа хранить как единое целое, нежели как набор файлов. Кроме того, архивация позволяет существенно уменьшить объём документа.

Архивы, помимо сжатия данных (а мы говорим только про сжатие **без потерь** и не касаемся, например, сжатия видео-информации), могут пригодиться для удобной упаковки разнородной информации в одном файле.

В Python с архивами можно работать разными способами:

— Как и в большинстве языков программирования, нам доступен запуск сторонних программ, и в этом случае мы можем просто вызвать стороннюю программу-архиватор;

— Стандартный модуль shutil (это мы уже немного попробовали);

— Стандартный модуль gzip;

— Стандартный модуль zipfile.

Мы остановимся на модуле **zipfile**, как наиболее полноценном и удобном.

Удобство заключается в том, что так как в архив может быть помещена целая структура каталогов, то работа с архивом подобна работе с обычными файлами и папками.

Напишем программу, которая выведет на экран содержание архива, который мы создали ранее, используя арсенал библиотеки **shutil**:

```
from zipfile import ZipFile

with ZipFile('archive.zip') as myzip:
    myzip.printdir()

-------------------------------------------------


    File Name                                             Modified             Size
    1/                                     2017-09-18 16:02:12            0
    2/                                     2017-09-18 16:02:14            0
    3/                                     2017-09-18 16:02:44            0
    1.txt                                  2017-09-18 15:59:30           35
                                           2017-09-18 14:28:20            0
    Копия.txt                              2017-09-18 16:26:52            0
                                           2017-09-18 16:02:12            0
                                           2017-09-18 16:02:14            0
    3/files/                               2017-09-18 16:02:30            0
                                           2017-09-18 16:02:16            0
    3/Описание.txt                         2017-09-18 16:02:40            0
    3/files/csvs/                          2017-09-18 16:02:58            0
                                  2017-09-18 16:02:36            0
    3/files/csvs/данные.csv                2017-09-18 16:02:52            0
```

В начале работы мы создаем объект типа **ZipFile**, передавая ему имя архива. Можно указать и необязательный параметр mode (режим работы), который принимает в себя значения r,w или a (всё по аналогии с «чистыми» файлами). Но по умолчанию считается, что архив открывается для чтения, поэтому мы не будем его указывать.

Кроме печати, можно получать информацию о файлах в архиве в виде списка:

```
with ZipFile('archive.zip') as myzip:
    info = myzip.infolist()
print(info[0].orig_filename)

-------------------------------------------------


    1/
```

А также имена файлов в архиве, тоже в виде списка:

```
with ZipFile('archive.zip') as myzip:
    print(myzip.namelist())
```

```
--------------------------------------------------

    ['1/', '2/', '3/', '1.txt', 'Icon\r', 'Копия.txt', '1/Icon\r',
     '2/Icon\r', '3/files/', '3/Icon\r', '3/Описание.txt',
     '3/files/csvs/', '3/files/csvs/Icon\r',
     '3/files/csvs/данные.csv']
```

Согласитесь, что такой способ очень похож на классический вариант работы с файлами, который мы рассматривали ранее.

Структуру архива мы получили, «вытащим» теперь и конкретный файл:

```
with ZipFile('archive.zip') as myzip:
    with myzip.open('1.txt', 'r') as file:
        print(file.read())

--------------------------------------------------

 b'\xd0\x9f\xd1\x80\xd0\xbe\xd0\xb8\xd0\xb7\xd0\xb2\xd0\xbe\xd0\xbb\xd1
\x8c\xd0\xbd\xd1\x8b\xd0\xb9 \xd1\x82\xd0\xb5\xd0\xba\xd1\x81\xd1\x82'
```

Что же у нас получилось? Обратите внимание на символ **b** перед выводом. Это бинарная последовательность. Но мы-то знаем, что перед нами — текстовый файл, поэтому мы можем быстро преобразовать (декодировать) эту строку. Надо только помнить, в какой кодировке записан файл.

```
with ZipFile('archive.zip') as myzip:
    with myzip.open('1.txt', 'r') as file:
        print(file.read().decode('utf-8'))

--------------------------------------------------

   Произвольный текст
```

По аналогии с чтением файлов из архива их можно туда и записывать.

```
with ZipFile('archive.zip', 'w') as myzip:
    myzip.write('test.txt')
    print(myzip.namelist())

--------------------------------------------------

    ['test.txt']
```

Сейчас мы создали новый архив, а предыдущий уничтожили, поэтому для добавления файла в архив поступим так:

```
with ZipFile('archive.zip', 'a') as myzip:
    myzip.write('test.txt')
    print(myzip.namelist())

--------------------------------------------------

    ['1/', '2/', '3/', '1.txt', 'Icon\r', 'Копия.txt', '1/Icon\r',
     '2/Icon\r', '3/files/', '3/Icon\r', '3/Описание.txt',
     '3/files/csvs/', '3/files/csvs/Icon\r',
     '3/files/csvs/данные.csv', 'test.txt']
```

Вот! Теперь другое дело!

Кроме того, у ZipFile есть метод **extractall**, который вытаскивает из архива всё содержимое в указанную папку. Интересный момент: если папку не указывать, то данные сложатся в «текущую папку», то есть в данном случае туда, где находится файл с программой. Структура каталогов при этом сохраняется.

```
ZipFile.extractall(path=None, members=None, pwd=None)
```

5. Формат JSON

JSON (англ. JavaScript Object Notation) — один из самых популярных типов структурированных файлов, поддерживающих произвольную вложенность. Это формат объекта в языке JavaScript, содержащий в себе сочетание словарей и списков (в терминах Python). Оказывается, что вместо того, чтобы передавать или хранить файлы в каких-то форматах, потом при помощи библиотек обрабатывая их, удобней передавать сами объекты языка. Ведь в этом случае не надо ничего дополнительно обрабатывать. Перед нами — уже готовый объект. Так как в последнее время очень распространились веб-приложения на JavaScript, JSON стал одним из самых популярных форматов, в том числе и в других языках.

Чтобы лучше понять JSON, давайте сначала посмотрим на модуль **pickle**. Он служит для того, чтобы превращать любой объект Python в байтовую структуру и обратно:

```
from pickle import loads, dumps
s = {'Иван': 24, 'Сергей': 11}
d = dumps(s)
```

```
print(d)

----------------------------------------------------

 b'\x80\x03}q\x00(X\x08\x00\x00\x00\xd0\x98\xd0\xb2\xd0\xb0\xd0\xbdq
\x01K\x18X\x0c\x00\x00\x00\xd0\xa1\xd0\xb5\xd1\x80\xd0\xb3\xd0\xb5
\xd0\xb9q\x02K\x0bu.'
```

```
loads(d)

----------------------------------------------------

{'Иван': 24, 'Сергей': 11}
```

Объект, представленный в виде массива байт, легко хранить на жёстком диске, в базах данных, передавать по сети между двумя процессами и т.д. Кстати, в некоторых нединамических языках очень недостаёт возможности сохранить объект на диск без дополнительного его преобразования. Это иногда приводит к таким странным вещам, как, например, в случае с интерпретатором C++, который сохраняет во вне и восстанавливает объекты этого языка во время работы программы (чего не может компилятор).

Теперь посмотрим, что же такое JSON.

Запросим по ссылке ссылке у Яндекс.Карт информацию о московском аэропорте Внуково.

Пройдите по этой ссылке и убедитесь, что ответ действительно приходит.

Нам вернётся вот такой ответ:

```
{
        "response": {
                "GeoObjectCollection": {
                        "metaDataProperty": {
                                "GeocoderResponseMetaData": {
                                        "request": "аэропорт Внуково",
                                        "found": "2",
                                        "results": "10"
                                }
                        },
                        "featureMember": [{
                                "GeoObject": {
                                        "metaDataProperty": {
                                                "GeocoderMetaData": {
                                                        "kind": "airport",
                                                        "text": "Россия, Москва,
```

```
                                "precision": "other",
                                "Address": {
                                        "country_code": "
                                        "formatted": "Moc
                                        "Components": [{
                                                "kind": "
                                                "name": "
                                        },
                                        {
                                                "kind": "
                                                "name": "
                                        },
                                        {
                                                "kind": "
                                                "name": "
                                        },
                                        {
                                                "kind": "
                                                "name": "
                                        }]
                                },
                                "AddressDetails": {
                                        "Country": {
                                                "AddressL
                                                "CountryN
                                                "CountryN
                                                "Administ
                                                        "
                                                        "
                                                                                        }
                                                }
                                        }
                                }
                        },
                        "description": "Москва, Россия",
                        "name": "аэропорт Внуково",
                        "boundedBy": {
                                "Envelope": {
                                        "lowerCorner": "37.229833
                                        "upperCorner": "37.303809
                                }
                        },
                        "Point": {
```

```
                                        "pos": "37.286292 55.605058"
                                }
                        }
                },
                {
                        "GeoObject": {
                                "metaDataProperty": {
                                        "GeocoderMetaData": {
                                                "kind": "railway",
                                                "text": "Россия, Киевское
                                                "precision": "other",
                                                "Address": {
                                                        "country_code": "
                                                        "formatted": "Кие
                                                        "Components": [{
                                                                "kind": "
                                                                "name": "
                                                        },
                                                        {
                                                                "kind": "
                                                                "name": "
                                                        },
                                                        {
                                                                "kind": "
                                                                "name": "
                                                        },
                                                        {
                                                                "kind": "
                                                                "name": "
                                                        }]
                                                },
                                                "AddressDetails": {
                                                        "Country": {
                                                                "AddressL
                                                                "CountryN
                                                                "CountryN
                                                                "Country"
                                                                        "
                                                                }
                                                        }
                                                }
                                        }
                                },
                                "description": "Киевское направление Моск
                                "name": "платформа Аэропорт Внуково",
                                "boundedBy": {
                                        "Envelope": {
```

```
                                                        "lowerCorner": "37.279914
                                                        "upperCorner": "37.296371
                                                }
                                        },
                                        "Point": {
                                                "pos": "37.288142 55.605765"
                                        }
                                }
                        }]
                }
        }
}
```


Как видно из этого примера, JSON очень похож по синтаксису на формат словаря в Python. Поэтому любая работа с JSON в Python происходит по алгоритму:

1. Создание словаря с данными.

2. Преобразование словаря в JSON-объект.

3. Передача данных.

Или в обратную сторону:

1. Получение файла или строки с JSON-содержимым.

2. Преобразование данных в словарь Python.

3. Работа с данными.

Для работы с JSON есть стандартный модуль **json**.

Поработаем с ним. Нам доступны уже знакомые функции **loads** (загрузка из строки JSON-объекта и преобразование его в Python-объект) и **dumps** (обратное преобразование).

```
import json

data = '''
{"response":{"GeoObjectCollection":{"metaDataProperty":
{"GeocoderResponseMetaData":{"request":"аэропорт Внуково",
"found":"2","results":"10"}},"featureMember":
[{"GeoObject":{"metaDataProperty":{"GeocoderMetaData":
{"kind":"airport","text":"Россия, Москва, аэропорт Внуково",
"precision":"other","Address":{"country_code":"RU",
"formatted":"Москва, аэропорт Внуково","Components":
[{"kind":"country","name":"Россия"},{"kind":"province",
"name":"Центральный федеральный округ"},
{"kind":"province","name":"Москва"},
```

{"kind":"airport","name":"аэропорт Внуково"}]},
"AddressDetails":{"Country":
{"AddressLine":"Москва, аэропорт Внуково",
"CountryNameCode":"RU","CountryName":"Россия",
"AdministrativeArea":
{"AdministrativeAreaName":"Москва",
"Locality":{"DependentLocality":
{"DependentLocalityName":"аэропорт Внуково"}}}}}},
"description":"Москва, Россия","name":"аэропорт Внуково",
"boundedBy":{"Envelope":
{"lowerCorner":"37.229833 55.583169",
"upperCorner":"37.303809 55.61598"}},
"Point":{"pos":"37.286292 55.605058"}}},
{"GeoObject":{"metaDataProperty":
{"GeocoderMetaData":{"kind":"railway",
"text":"Россия, Киевское направление Московской железной дороги,
платформа Аэропорт Внуково","precision":"other",
"Address":{"country_code":"RU",
"formatted":"Киевское направление Московской железной дороги,
платформа Аэропорт Внуково",
"Components":[{"kind":"country","name":"Россия"},
{"kind":"province","name":"Центральный федеральный округ"},
{"kind":"route",
"name":"Киевское направление Московской железной дороги"},
{"kind":"railway","name":"платформа Аэропорт Внуково"}]},
"AddressDetails":{"Country":
{"AddressLine":"Киевское направление Московской железной дороги,
платформа Аэропорт Внуково","CountryNameCode":"RU",
"CountryName":"Россия","Country":{"Locality":""}}}}},
"description":"Киевское направление Московской железной дороги, Россия",
"name":"платформа Аэропорт Внуково","boundedBy":{"Envelope":
{"lowerCorner":"37.279914 55.601106",
"upperCorner":"37.296371 55.610423"}},
"Point":{"pos":"37.288142 55.605765"}}}]}}}
'''

resp = json.loads(data)
print(resp['GeoObjectCollection']['featureMember']
          ['GeoObject']['metaDataProperty']['GeocoderMetaData']['kind'])

----

airport

```
s = {'Иван': 24, 'Сергей': 11}
json.dumps(s)

-----------------------------------------------------

'{"\\u0418\\u0432\\u0430\\u043d": 24, "\\u0421\\u0435\\u0440\\u0433
   \\u0435\\u0439": 11}'
```

Распространение и популярность JSON привели к тому, что он используется в **API** (англ. Application Programming Interface) многих сервисов (социальные сети, научные проекты), а также в некоторых базах данных, например, **MongoDB** в качестве базового формата для документов.

Но о работе с API мы поговорим чуть позже.