Урок json, xml

Работа с файловой системой и популярными форматами файлов: zip-архивами и json-файлами

Аннотация

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

1. Введение

Мы уже научились создавать программы с графическим пользовательским интерфейсом с использованием виджетов PyQt, работать с базами данных, а также создавать игровые приложения с применением библиотеки PyGame. В этом полугодии мы будем рассматривать еще одну сложную и многообразную тему — работу с сетью Интернет. Периодически мы будем немного отвлекаться непосредственно от самого Интернета и говорить о смежных темах, которые сильно облегчат нам жизнь в дальнейшем.

2. Манипуляции с файлами и папками

Когда мы создавали PyQt-приложения, мы говорили о том, что такое файл, какими они бывают и как с каждым типом файлов работать с помощью языка программирования Python. Давайте копнем немного глубже в этом направлении.

Уверены, что ни для кого не является сюрпризом, что файлы в современных операционных системах (ОС)

не валяются в одной куче, а организованы особым образом. Почти все ОС поддерживают технологию иерархической, или древовидной, организации файловых систем. Кроме понятия **файл** существует также и понятие **папка** (или каталог). Папка — это контейнер, содержащий файлы, причем папка может быть вложена в другую папку, а та — в другую папку и т. д. Файловая система представляет собой своеобразное дерево с вершинами-папками и листьями-файлами. В UNIX-подобных операционных системах благодаря концепции «все 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 содержимым файла, в Python есть средства для работы 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 начнем обзор возможностей с модуля os, потом рассмотрим модуль os.path и модуль shutil.

3. Модули os и os.path

Модуль os содержит в себе некоторые уникальные функции, зависящие от конкретной операционной системы. Посмотрим, что же в нем находится. Нам поможет функция `dir()`.

```
import os

print(dir(os))
```

```
['DirEntry', 'F_OK', 'MutableMapping', 'O_APPEND', 'O_BINARY', 'O_CREAT',
 'O_EXCL', 'O_NOINHERIT', 'O_RANDOM', 'O_RDONLY', 'O_RDWR', 'O_SEQUENTIAL',
 'O_SHORT_LIVED', 'O_TEMPORARY', 'O_TEXT', 'O_TRUNC', 'O_WRONLY',
 'P_DETACH', 'P_NOWAIT', 'P_NOWAITO', 'P_OVERLAY', 'P_WAIT', 'PathLike',
 'R_OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_SET', 'TMP_MAX', 'W_OK', 'X_OK',
 '_Environ', '__all__', '__builtins__', '__cached__', '__doc__',
 '__file__', '__loader__', '__name__', '__package__', '__spec__',
 '_execvpe', '_exists', '_exit', '_fspath', '_get_exports_list', '_putenv',
 '_unsetenv', '_wrap_close', 'abc', 'abort', 'access', 'altsep', 'chdir',
 'chmod', 'close', 'closerange', 'cpu_count', 'curdir', 'defpath',
 'device_encoding', 'devnull', 'dup', 'dup2', 'environ', 'error', 'execl',
 'execle', 'execlp', 'execlpe', 'execv', 'execve', 'execvp', 'execvpe',
 'extsep', 'fdopen', 'fsdecode', 'fsencode', 'fspath', 'fstat', 'fsync',
 'ftruncate', 'get_exec_path', 'get_handle_inheritable', 'get_inheritable',
 'get_terminal_size', 'getcwd', 'getcwdb', 'getenv', 'getlogin', 'getpid',
 'getppid', 'isatty', 'kill', 'linesep', 'link', 'listdir', 'lseek', 'lstat',
 'makedirs', 'mkdir', 'name', 'open', 'pardir', 'path', 'pathsep', 'pipe',
 'popen', 'putenv', 'read', 'readlink', 'remove', 'removedirs', 'rename',
 'renames', 'replace', 'rmdir', 'scandir', 'sep', 'set_handle_inheritable',
```

```
 'set_inheritable', 'spawnl', 'spawnle', 'spawnv', 'spawnve', 'st',
 'startfile', 'stat', 'stat_result', 'statvfs_result', 'strerror',
 'supports_bytes_environ', 'supports_dir_fd', 'supports_effective_ids',
 'supports_fd', 'supports_follow_symlinks', 'symlink', 'sys', 'system',
 'terminal_size', 'times', 'times_result', 'truncate', 'umask', 'uname_result',
 'unlink', 'urandom', 'utime', 'waitpid', 'walk', 'write']
```

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

Аттрибут `os.name` содержит строковое значение с типом операционной системы, в которой выполняется наша программа. Допустимые значения:

- `posix` — для linux и macOS
- `nt` — для операционных систем семейства Windows
- `java` — для систем, работающих в виртуальной Java-машине (например, Android)

```
import os

print(os.name)
```

```
posix
```

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

```
import os

print(os.getcwd())
```

```
/SomeUser/some folder
```

Если текущий каталог не был изменен (об этом ниже), то функция `os.getcwd()` вернет адрес каталога, в котором лежит ваша программа.

Для смены текущего каталога используется функция `os.chdir()`:

```
import os

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

```
/SomeUser/some folder/files
```

Для чего это может быть нужно? Давайте, например, представим, что у нас идет обработка большого количества файлов, которые лежат в папке с нашей программой, в директории files. Тогда из программы необходимо будет обращаться к этим файлам с указанием относительного пути: files/file1, files/file2 и т. д. Это не всегда удобно. Иногда значительно проще поменять текущий каталог на /files и обращаться к файлам непосредственно по имени: file1, file2 и т. д.

Для того чтобы перейти из текущего каталога в родительский, необходимо вызвать функцию `os.chdir()`

с аргументом '..'

```
import os

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

```
/SomeUser/
```

Можно использовать не только относительные пути, но и абсолютные, например: C:\Program files.

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

```
os.access("1.txt", os.F_OK)
```

```
True
```

```
os.access("1.txt", os.R_OK)
```

```
True
```

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

```
False
```

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

Для получения списка файлов и вложенных каталогов используется функция os.listdir():

```
import os

print(os.listdir())
```

```
['1', '2', '3', '1.txt', 'Icon\r']
```

В качестве параметра в os.listdir() можно передать относительный или абсолютный адрес каталога. Если ничего не передать, то напечатается список файлов и директорий в текущей папке.

Очень полезной может быть функция рекурсивного прохода по всем папкам в заданной папке (примерно такой же функциональности можно добиться самостоятельно, написав рекурсивный алгоритм, с использованием функции os.listdir() и os.chdir()).

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']
```

Видим, что функция `os.walk()` возвращает кортеж из трех элементов: текущий каталог, список вложенных каталогов, список файлов текущей директории.

Рекурсивно перебирая папки, мы всегда имеем 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))
```

```
['__all__', '__builtins__', '__cached__', '__doc__', '__file__',
 '__loader__', '__name__', '__package__', '__spec__', '_get_sep',
 '_joinrealpath', '_varprog', '_varprogb', 'abspath', 'altsep',
 'basename', 'commonpath', 'commonprefix', 'curdir', 'defpath',
 'devnull', 'dirname', 'exists', 'expanduser', 'expandvars',
 'extsep', 'genericpath', 'getatime', 'getctime', 'getmtime',
 'getsize', 'isabs', 'isdir', 'isfile', 'islink', 'ismount', 'join',
 'lexists', 'normcase', 'normpath', 'os', 'pardir', 'pathsep',
 'realpath', 'relpath', 'samefile', 'sameopenfile', 'samestat',
 'sep', 'split', 'splitdrive', 'splitext', 'stat',
 'supports_unicode_filenames', 'sys']
```

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 функциональность различных библиотек.

```
os.path.exists('files/3')
```

```
True
```

```
os.path.isfile('files/3')
```

```
False
```

```
os.path.isdir('files/2')
```

```
True
```

Функция `os.path.abspath()` вернет абсолютный путь по относительному:

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

```
'/SomeUser/some folder/1.txt'
```

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

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

```
''
```

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

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

4. Модуль shutil (shell utilities)

Еще один полезный модуль — shutil (правда веселое название?). В нем содержатся несколько высокоуровневых функций для манипуляции с файлами: копирование, перемещение, удаление и другие. Все 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)
- Приемник (куда копируем)

Оба параметра — строки.

```
import shutil

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

```
'files/Копия.txt'
```

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

Удаление папки со всем ее содержимым выполняется функцией `shutil.rmtree()` (от английского remove tree). В качестве параметра передается 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')
```

```
'/SomeUser/some folder/archive.zip'
```

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

5. Zip-архивы

Раз уж мы заговорили об архивах, 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 распространенный 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 (даже среди тех форматов, с которыми мы уже работали), 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 объем 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 функциональном и удобном.

Основное преимущество данной библиотеки заключается в том, что она позволяет работать с архивом (а в архив может быть помещена целая структура каталогов), как с обычной папкой, содержащей файлы и другие каталоги.

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` (все 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, 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\x
```

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 уже существующий архив будем работать с ним с ключом a:

```
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 все 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)
```

6. Формат JSON

Вот мы и добрались до, пожалуй, самого популярного формата файлов в Интернете — JSON. Некоторые из вас могут удивиться этому утверждению, так как услышат про этот формат впервые. На самом деле все из вас пользовались данным форматом, пусть и неявно. Именно JSON обмениваются большинство приложений в Интернете.

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\x
```

```
loads(d)
```

```
{'Иван': 24, 'Сергей': 11}
```

Объект, представленный в виде массива байт, легко хранить на жестком диске, в базах данных, передавать по сети между двумя приложениями или от одного приложения другому в рамках одного компьютера и т. д.

Теперь посмотрим, что же такое JSON.

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

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

Нам вернется вот такой ответ:

```
{
        "response": {
                "GeoObjectCollection": {
                        "metaDataProperty": {
```

```
                "GeocoderResponseMetaData": {
                        "request": "аэропорт Внуково",
                        "found": "2",
                        "results": "10"
                }
        },
        "featureMember": [{
                "GeoObject": {
                        "metaDataProperty": {
                                "GeocoderMetaData": {
                                        "kind": "airport",
                                        "text": "Россия, Москва, аэропорт Внуко
                                        "precision": "other",
                                        "Address": {
                                                "country_code": "RU",
                                                "formatted": "Москва, аэропорт
                                                "Components": [{
                                                        "kind": "country",
                                                        "name": "Россия"
                                                },
                                                {
                                                        "kind": "province",
                                                        "name": "Центральный фе
                                                },
                                                {
                                                        "kind": "province",
                                                        "name": "Москва"
                                                },
                                                {
                                                        "kind": "airport",
                                                        "name": "аэропорт Внуко
                                                }]
                                        },
                                        "AddressDetails": {
                                                "Country": {
                                                        "AddressLine": "Москва,
                                                        "CountryNameCode": "RU"
                                                        "CountryName": "Россия"
                                                        "AdministrativeArea": {
                                                                "Administrative
                                                                "Locality": {
                                                                        "Depend

                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                }
```

```
                        },
                        "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 М
                                        "precision": "other",
                                        "Address": {
                                                "country_code": "RU",
                                                "formatted": "Киевское направле
                                                "Components": [{
                                                        "kind": "country",
                                                        "name": "Россия"
                                                },
                                                {
                                                        "kind": "province",
                                                        "name": "Центральный фе
                                                },
                                                {
                                                        "kind": "route",
                                                        "name": "Киевское напра
                                                },
                                                {
                                                        "kind": "railway",
                                                        "name": "платформа Аэро
                                                }]
                                        },
                                        "AddressDetails": {
                                                "Country": {
                                                        "AddressLine": "Киевско
                                                        "CountryNameCode": "RU"
                                                        "CountryName": "Россия"
                                                        "Country": {
                                                                "Locality": ""
                                                        }
```

```
                                                        }
                                                }
                                        }
                                },
                                "description": "Киевское направление Московской железно
                                "name": "платформа Аэропорт Внуково",
                                "boundedBy": {
                                        "Envelope": {
                                                "lowerCorner": "37.279914 55.601106",
                                                "upperCorner": "37.296371 55.610423"
                                        }
                                },
                                "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. Получение 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

7. Библиотека для работы с JSON

Для работы с JSON есть стандартный модуль json.

Поработаем с ним. Этот модуль содержит аналогичный `pickle` интерфейс, то есть нам доступны функции `loads()` (загрузка из строки JSON-объекта и преобразование его в Python-объект) и `dumps()` (обратное преобразование).

Чтение json.

Откроем в IDE файл **cats_json.json**. Мы видим, что там в виде словаря записана некоторая информация о питомце:


```
{
  "name": "Barsik",
  "age": 7,
```

```
  "meals": [
    "Wiskas",
    "Royal Canin",
    "Purina",
    "Hills",
    "Brit Care"
  ]
}
```

Обратите внимание: в формате JSON используются только двойные кавычки.

Для чтения данных в модуле `json` есть два метода:

1. `json.load()` — считывает целиком файл в формате JSON и возвращает объекты Python
2. `json.loads()` — считывает строку в формате JSON и возвращает объекты Python

Напишем код, читающий файл и выводящий содержимое полученного словаря:

```
import json

with open('cats_json.json') as cat_file:
    data = json.load(cat_file)
for key, value in data.items():
    if type(value) == list:
        print(f'{key}: {", ".join(value)}')
    else:
        print(f'{key}: {value}')
```

```
name: Barsik
age: 7
meals: Wiskas, Royal Canin, Purina, Hills, Brit Care
```

В отличие от метода `json.load(filename)`, метод `json.loads(string)` считывает строку и возвращает объект json, если его возможно получить из переданной строки.

В случае, если строка или файловый объект содержат некорректный JSON, будет выброшено исключение типа json.decoder.JSONDecodeError.

```
import json

s = 'Не Json'
json.loads(s)
```

```
json.decoder.JSONDecodeError: Expecting value: line 1 column 1 (char 0)
```

Создайте в IDE новый файл cats_2.json, в который кроме Барсика добавьте еще одного кота по вашему усмотрению. Поскольку теперь в файле у нас уже два словаря, их надо объединить в одну структуру, пусть это будет список. Получим что-то вроде такого содержимого:

```
[
  {
```

```
    "name": "Barsik",
    "age": 7,
    "meals": [
      "Wiskas",
      "Royal Canin",
      "Purina",
      "Hills",
      "Brit Care"
    ]
  },
  {
    "name": "Mursik",
    "age": 3,
    "meals": [
      "Purina",
      "Hills",
      "Brit Care"
    ]
  }
]
```

Чтобы воспользоваться методом `loads()`, нужно сначала считать весь файл в строку, а затем передать ее методу для преобразования в json-объект.

```
import json

with open('cats_2.json') as cat_file:
    f = cat_file.read()
    data = json.loads(f)
    for item in data:
        for key, value in item.items():
            if type(value) == list:
                print(f'{key}: {", ".join(value)}')
            else:
                print(f'{key}: {value}')
        print()
```

И в том, и в другом случае мы получили объект языка Python, словарь или список, с которым можно сразу работать средствами языка.

В таблице представлено соответствие между данными в Python и в JSON:

Python	JSON
dict	object
list, tuple	array
str	string
int, float	number
True	true

Python	JSON
False	false
None	null

При обратном преобразовании массив `array` преобразуется в список.

Запись в файл.

Для записи информации в файл в `json` также есть два метода:

1. `json.dump()` — метод записывает объект Python в файл в формате JSON
2. `json.dumps()` — метод преобразует объект Python в строку в формате JSON

Давайте создадим словарь, в котором коту добавим хозяев, а затем полученную информацию сохраним в файле:

```
import json

cats_dict = {
    'name': 'Pushin',
    'age': 1,
    'meals': [
        'Purina', 'Cat Chow', 'Hills'
    ],
    'owners': [
        {
            'first_name': 'Bill',
            'last_name': 'Gates'
        },
        {
            'first_name': 'Melinda',
            'last_name': 'Gates'
        }
    ]
}

with open('cats_3.json', 'w') as cat_file:
    json.dump(cats_dict, cat_file)
```

Если сейчас открыть в IDE PyCharm файл cats_3.json, который был создан нашей программой, мы увидим, что json выведен в одну строку без форматирования. Для представления json в удобном для чтения виде в PyCharm можно использовать горячую клавишу Ctrl-Alt-L.

Метод `dumps()` используется, когда надо просто преобразовать объект в json-формат, необязательно для записи в файл. Это нужно, например, при передаче информации в веб-приложении.

Посмотрим на примере:

```
import json

weekdays = {i: day
            for i, day in enumerate(['Sunday',
                                     'Monday',
                                     'Tuesday',
```

```
                                    'Wednesday',
                                    'Thursday',
                                    'Friday',
                                    'Saturday'
                                    ])}
data = json.dumps(weekdays)
print(data)
print(type(data))
```

Получим строку в формате json:

```
{"0": "Sunday", "1": "Monday", "2": "Tuesday", "3": "Wednesday", "4": "Thursday", "5": "Friday",
<class 'str'>
```

Дополнительные параметры методов записи.

Методы записи имеют несколько необязательных параметров, которые можно менять для более удобного чтения человеком. Рассмотрим некоторые из них.

ensure_ascii. В случае, если `ensure_ascii=True` (по умолчанию), все не-ASCII символы в выводе будут экранированы последовательностями \uXXXX. Если `ensure_ascii=False`, эти символы будут записаны как есть. Это важно, если в содержимом есть русские буквы.

Запишем в файл cats.json содержимое списка без флага:

```
with open('cats.json', 'w') as file:
    json.dump(data, file)
```

В файле:

```
[
    {
        "name": "\u0411\u0430\u0440\u0441\u0438\u043a",
        "age": 7,
        "toys": [
            "\u041c\u044b\u0448\u043a\u0430",
            "\u041f\u0440\u0443\u0442\u0438\u043a",
            "\u0411\u0430\u043d\u0442\u0438\u043a",
            "\u0421\u0432\u043e\u0439 \u0445\u0432\u043e\u0441\u0442"
        ]
    },
    {
        "name": "\u041c\u0443\u0440\u0437\u0438\u043a",
        "age": 3,
        "toys": [
            "\u0420\u0443\u043a\u0430\u0432 \u0445\u043e\u0437\u044f\u0439\u043a\u0438",
            "\u0428\u043d\u0443\u0440 \u043e\u0442 \u0442\u0435\u043b\u0435\u0432\u0438\u0437\u0
            "\u041e\u0431\u043e\u0438 \u043d\u0430 \u0441\u0442\u0435\u043d\u0435"
        ]
    }
]
```

А теперь — с флагом:

```
with open('cats.json', 'w') as file:
    json.dump(data, file, ensure_ascii=False)
```

В файле:

```
[
    {
        "name": "Барсик",
        "age": 7,
        "toys": [
            "Мышка",
            "Прутик",
            "Бантик",
            "Свой хвост"
        ]
    },
    {
        "name": "Мурзик",
        "age": 3,
        "toys": [
            "Рука хозяйки",
            "Шнур от телевизора",
            "Обои на стене"
        ]
    }
]
```

indent. Отступ `indent` нужен для удобного для чтения человеком представления информации в объекте JSON. По умолчанию имеет значение None для более компактного представления, то есть без отступов. Также отступов не будет, если значение `indent` равно 0, отрицательному числу или пустой строке. Если `indent` — строка (например, `\t`), эта строка используется в качестве отступа.

Пример со значением `indent=2`:

```
with open('cats.json', 'w') as file:
    json.dump(data, file, ensure_ascii=False, indent=2)
```

Результат:

```
[
  {
    "name": "Барсик",
    "age": 7,
    "toys": [
      "Мышка",
      "Прутик",
      "Бантик",
      "Свой хвост"
```

```
    ]
  },
  {
    "name": "Мурзик",
    "age": 3,
    "toys": [
      "Рука хозяйки",
      "Шнур от телевизора",
      "Обои на стене"
    ]
  }
]
```

sort_keys. Если `sort_keys=True` (по умолчанию False), ключи выводимого словаря будут отсортированы. Это удобно, если ключей много.

Пример со значением `sort_keys=True`:

```
with open('cats.json', 'w') as file:
    json.dump(data, file, ensure_ascii=False,
              indent=2, sort_keys=True)
```

Результат:

```
[
  {
    "age": 7,
    "name": "Барсик",
    "toys": [
      "Мышка",
      "Прутик",
      "Бантик",
      "Свой хвост"
    ]
  },
  {
    "age": 3,
    "name": "Мурзик",
    "toys": [
      "Рука хозяйки",
      "Шнур от телевизора",
      "Обои на стене"
    ]
  }
]
```

8. В заключении

Сериализация и десериализация

Мы производили преобразования между объектами языка Python и json-объектами. Такие преобразования называются **сериализацией** (кодирование в json-формат, то есть в поток байт) и **десериализацией** (декодирование в объект языка).

Есть еще несколько моментов при работе с JSON, о которых стоит помнить:

1. Чтобы не возникали проблемы с кодировкой, если в файл передаются данные с русскими буквами, как и во всех других случая работы с файлом, при открытии нужно принудительно устанавливать кодировку (особенно актуально для OS семейства Windows):

```
with open('cats_3.json', 'w', encoding='utf8') as cat_file:
    cat_file.write(json.dumps(cats_dict))
```

2. При создании json-файла «вручную» нужно помнить, что в нем нельзя использовать одинарные кавычки. При создании программными средствами нужные кавычки ставятся автоматически.

3. Ключами словаря в json не могут быть кортежи и числа. Но ключ-число не вызовет ошибку при сериализации, он будет просто преобразован в строку.

4. Помните, что при преобразовании данные будут не всегда того же типа, что были в Python, то есть:

```
print(json.loads(json.dumps(weekdays)) == weekdays)  # False
```

Справка

Более подробную информацию по работе с модулем `json` можно найти по **ссылке**.

О работе непосредственно с API мы поговорим уже на следующем уроке.