

Неделя 3. Часть 1. IO, datetime в Python

Цели лекции

- научиться работать с файлами различных форматов;
- научиться работать с модулем datetime.

План лекции

План лекции	1
Input-output в Python	2
ЗАПИСЬ И ЧТЕНИЕ ФАЙЛОВ.....	2
Примеры, взятые с видео-лекции:	2
Управление курсором.....	3
РЕЖИМЫ РАБОТЫ С ФАЙЛАМИ. ОСОБЕННОСТИ	3
МЕТОДЫ ОБЪЕКТОВ-ФАЙЛОВ	4
ФУНКЦИИ REPR() И EVAL()	4
ТЕКСТОВЫЕ ФОРМАТЫ ФАЙЛОВ	5
JSON	5
CSV	5
Модуль Datetime	6
ВВЕДЕНИЕ В DATETIME	6
МАНИПУЛИРОВАНИЕ ДАТАМИ	8
ПРЕОБРАЗОВАНИЯ	8
ПОЛЕЗНЫЕ ССЫЛКИ	9

Input-output в Python

ЗАПИСЬ И ЧТЕНИЕ ФАЙЛОВ

Функция `open()` возвращает объект файла и в большинстве случаев используется с двумя аргументами:
`open(имя_файла, режим)`.

```
>>> f = open('filename.txt', 'w')
```

Создан `f` - файл-поинтер (файловый объект- указатель на файл)

Первый параметр — строка, содержащая путь к файлу.

Второй — строка, содержащая несколько символов, описывающих способ использования файла (**параметр режима, модификатор, дескриптор**). Значение **параметра режима** может быть:

'r' — если файл будет открыт только для чтения,

'w' — открыт только для записи (существующий файл с таким же именем будет стёрт; если файла не существует — он будет создан, что возможно только в режиме 'w'),

'a' — файл открыт для добавления: любые данные, записанные в файл автоматически добавляются в конец,

'r+' — открывает файл и для чтения, и для записи.

Параметр режима необязателен: если он опущен — предполагается, что он равен 'r'.

Функция записи в файл: `имя_файла.write(записываемая_строка)`.

После окончания работы с файлом его необходимо закрыть при помощи команды: `имя_файла.close()`.

Примеры, взятые с видео-лекции:

1. Открытие файла (создание) **для записи**, запись, закрытие файла

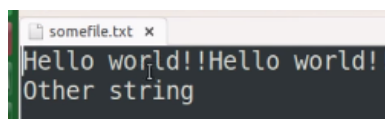
```
fp = open('somefile.txt', 'w')
fp.write("Hello world!!")
fp.close()
```

2. Открытие **для чтения**, чтение, закрытие (обратить внимание: при повторном вызове команды `read` получаем пустую строку; функция `read` возвращает все содержимое файла одной строкой; может принимать необязательный параметр — количество символов, которые необходимо прочесть):

```
fp = open('somefile.txt', 'r')
fp.read()
'Hello world!!Hello world!!'
fp.read()
''
fp.close()
```

3. Открытие файла для записи данных в конец файла:

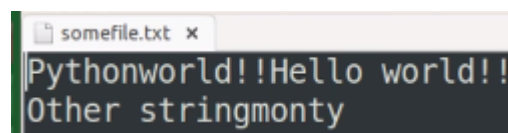
```
In [14]: fp = open('somefile.txt', 'a')
In [15]:
In [15]: fp.write("\nOther string")
In [16]:
In [16]:
In [16]: fp.close()
```



(содержимое файла)

4. Открытие файла **и для записи, и для чтения** (обратить внимание: при записи данные записываются в начало файла; при последующем чтении записанные только что данные не прочтены, т.к. после записи **курсор** (текущая позиция) установлен уже на символе, следующем за последним символом записанной строки):

```
In [20]: fp = open('somefile.txt', 'r+')
In [21]: fp.write("Python")
In [22]: fp.read()
Out[22]: 'world!!Hello world!!\nOther string'
In [23]: fp.write("monty")
In [24]: fp.close()
```



(в результате выполнения всех команд)

Управление курсором

`f.seek(смещение, откуда)` — изменяет позицию курсора. Позиция вычисляется прибавлением смещения к точке отсчёта; точка отсчёта выбирается из параметра откуда. Значение 0 параметра откуда отмеряет смещение от начала файла, значение 1 применяет текущую позицию в файле, а значение 2 в качестве точки отсчёта использует конец файла. Параметр откуда может быть опущен и по умолчанию устанавливается в 0, используя начало файла в качестве точки отсчёта. Если функция принимает один параметр, ...

`f.tell()` — возвращает целое число, представляющее собой текущую позицию в файле.

РЕЖИМЫ РАБОТЫ С ФАЙЛАМИ. ОСОБЕННОСТИ

Основных режимов работы с файлами 3: 'r', 'w', 'a' - чтение, запись, дозапись.

Только режим 'w' - создает файл на первом вызове.

Если вы хотите воспользоваться режимом 'r' – открыть для чтения – то в этом такой файл должен уже существовать. Та же ситуация в случае дозаписи – режим 'a'

Если файла не существует: в режимах 'r' и 'a' - возникает ошибка - файл не найден `FileNotFoundError`.

Рассмотрим следующий пример:

```
>>>f = open('filename.txt', 'w') # создается текстовый файл
>>>f = write('Some data')
```

```
# запись в файл не происходит мгновенно, данные в файл сбрасываются либо в момент
закрытия файла f.close() #или второй вариант функция f.flush() позволяет освободить
память
```

Если попытаться прочитать что-то из файла, который открыт для записи (дескриптор 'w') возникнет ошибка `UnsupportedOperation: not readable`.

```
>>> f.seek(0) # переводим курсор в начальную позицию файла
>>> f = write('Hello world')
# запись произведется поверх данных, поскольку курсор стоял не в конце
```

Если мы снова откроем этот файл в режиме записи, то данные будут обнулены – курсор стоит в начале файла.

Если же открыть файл в режиме дозаписи (дескриптор 'a'), то новые данные будут добавляться в конец файла

В режиме 'r+' файл не создается, но мы можем открыть в этом режиме существующий файл, при этом с правом читать из файла и писать в него.

В режиме 'a+' файл создается. И при этом нам доступно и чтение и запись: самый универсальный режим для небольших задач.

МЕТОДЫ ОБЪЕКТОВ-ФАЙЛОВ

В примерах ниже подразумевается, что заранее создан **файловый объект** (файл – тоже объект!) с именем `f`. `f.read(размер)` — функция читает некоторое количество данных и возвращает их в виде строки, размер — необязательный числовой параметр. Если размер опущен, будет прочитано и возвращено всё содержимое файла.

`f.readline()` — читает одну строку из файла начиная с курсора (текущей позиции); возвращает все содержимое файла в виде одной строки.

```
In [26]: print fp.readlines()
['Hello!\n', 'Hello world!\n', 'Hello Python!\n', 'Hello Student
!\n', '\n']
```

`f.readlines()` — считывает всё содержимое файла, и возвращает список строк.

`f.write(s)` — записывает содержимое строки `s` в файл; перенос строки нужно указывать в явном виде.

`f.writelines(lis)` — записывает содержимое списка строк `lis` в файл; перенос строки нужно указывать в явном виде.

`f.close()` — закрывает файл. При попытке использовать закрытый файл для операций чтения/записи генерируется исключение `ValueError`

ФУНКЦИИ `repr()` И `eval()`

Если мы хотим записать в файл более сложные структуры данных , например, словарь , то мы можем столкнуться с серьезными трудностями.

Воспользуемся функцией `repr()` , которая преобразует словарь в строку. Затем записываем в файл эту строку, которая выглядит как словарь. После чтения из словаря мы прочитаем строку. Чтобы эту строку превратить обратно в словарь, воспользуйтесь функцией `eval()` .

JSON

json формат - это простое преобразование структурированных типов данных в строку. Эту строку можно записать в файл и потом прочитать из файла. Выполнив обратное преобразование получим исходные данные.

Традиционно json используется для передачи форматированных данных в javascript

Преобразование исходного словаря в последовательность и запись в файл с расширением .json:

```
In [55]: fp = open('file.json', 'w')

In [56]: j = {'1': "2", '3': [1,2,3]}

In [57]: import json

In [58]: json.dump
json.dump  json.dumps

In [58]: json.dumps(j)
Out[58]: '{"1": "2", "3": [1, 2, 3]}'

In [59]: js = json.dumps(j)

In [61]: fp.write(js)

In [62]: fp.close()
```

Чтение файла.json (в итоге получаем d — исходный словарь):

```
In [63]: fp = open('file.json', 'r')

In [64]: j_in = fp.read
fp.read  fp.readinto  fp.readline  fp.readlines

In [64]: j_in = fp.read()

In [65]: j_in
Out[65]: '{"1": "2", "3": [1, 2, 3]}\n'

In [66]: d = json.loads(j_in)

In [67]: d
Out[67]: {'1': '2', '3': [1, 2, 3]}

In [68]: type(d)
Out[68]: dict
```

Подробнее про json: <https://docs.python.org/2/library/json.html#module-json>

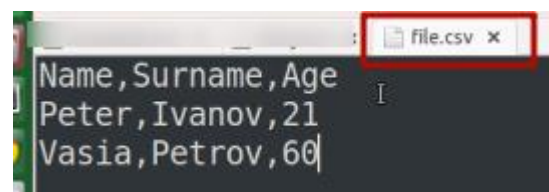
CSV

Файл в формате **CSV** (comma-separated values - значения, разделенные запятыми) - универсальное средство для переноса **табличной** информации между приложениями. Формат описывает набор записей таблицы, в котором каждая ячейка ряда отделена друг от друга запятой (**делимитер** по умолчанию— символ запятой), а каждая ряд символом переноса строки.

Каждая запись файла.csv, разделенная запятой (по умолчанию), будет представлена ячейкой.

Запись в csv-файл:

```
In [69]: import csv
In [70]: fp = open('file.csv', 'w')
In [71]: fp_csv = csv.writer(fp)
In [72]: fp_csv.w
fp_csv.writerow fp_csv.writerows
In [72]: fp_csv.writerow(["Name", "Surname", "Age"])
In [73]: fp_csv.writerow(["Peter", "Ivanov", "21"])
In [74]: fp_csv.writerow(["Vasia", "Petrov", "60"])
In [75]: fp.close()
```



(содержимое файла.csv)

Чтение из csv-файла:

(next возвращает следующую строку; ведет себя как итератор)

```
In [76]: fp = open('file.csv', 'r')
In [77]: fp_csv = csv.reader(fp)
In [78]: fp
fp      fp_csv
In [78]: fp_csv.
fp_csv.dialect fp_csv.line_num fp_csv.next
In [78]: fp_csv.next()
Out[78]: ['Name', 'Surname', 'Age']
In [79]: fp_csv.next()
Out[79]: ['Peter', 'Ivanov', '21']
```

Листинг с итератором:

```
In [84]: for iten in fp_csv:
        print iten
.....:
['Peter', 'Ivanov', '21']
['Vasia', 'Petrov', '60']
```

Доки и допинфа:

<https://docs.python.org/3/library/csv.html#module-csv>

<http://stackoverflow.com/questions/16823695/how-to-use-delimiter-for-csv-in-python>

<http://stackoverflow.com/questions/16312104/python-import-csv-file-delimiter-or>

Модуль Datetime

ВВЕДЕНИЕ В DATETIME

Существует множество модулей и библиотек для работы с датами (time, mxDateTime, date, pytz, arrow...). В рамках данного курса мы будем работать с относительным временем (**naive objects**), с модулем **datetime**.

Модуль datetime предоставляет классы для обработки времени и даты разными способами.

КЛАСС DATETIME.DATETIME

Объекты класса datetime используются для представления даты и времени.

Справка о классе:

```
>>> dir(datetime)
```

```
['__add__', '__class__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__gt__',  
 '__hash__', '__init__', '__le__', '__lt__', '__ne__', '__new__', '__radd__', '__reduce__', '__reduce_ex__', '__repr__',  
 '__rsub__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', 'astimezone', 'combine', 'ctime', 'date',  
 'day', 'dst', 'fromordinal', 'fromtimestamp', 'hour', 'isocalendar', 'isoformat', 'isoweekday', 'max', 'microsecond', 'min',  
 'minute', 'month', 'now', 'replace', 'resolution', 'second', 'strftime', 'strptime', 'time', 'timetuple', 'timetz', 'today',  
 'toordinal', 'tzinfo', 'tzname', 'utcfromtimestamp', 'utcnow', 'utcoffset', 'utctimetuple', 'weekday', 'year']
```

Конструктор datetime

`datetime.datetime(year, month, day[, hour[, minute[, second[, microsecond[, tzinfo]]]])` - конструктор класса. Первые 3 параметра (год, месяц и день) являются обязательными, другие необязательные и могут задаваться как именованные аргументы.

```
>>> print datetime.datetime(2015, 2, 20)
```

```
2015-02-20 00:00:00
```

```
>>> print(datetime.datetime(2015, 2, 20, minute=34))
```

```
2015-02-20 00:34:00
```

`datetime.datetime.today()` - метод класса, используется для создания объекта сегодняшней даты.

```
>>> print(datetime.datetime.today())
```

```
2015-03-05 12:56:15.646968
```

Форматирование даты

`datetime.datetime.strptime(date_string, format)` - метод класса, создает объект datetime из строки date_string, считая, что он содержит дату в формате format.

```
>>> print datetime.datetime.strptime('2015-03-22', '%Y-%m-%d')
```

```
2015-03-22 00:00:00
```

Поля объектов класса доступны только для чтения:

```
>>> x = datetime.datetime.today()
```

```
>>> print x
```

```
2015-03-05 13:04:20.611847
```

Атрибуты объекта:

`x.year` - год (целое число). 2015

`x.month` - месяц (целое число от 1 до 12). 3

`x.day` - число (целое число от 1 до количества дней в месяце). 5

`x.hour` - часы (целое число от 0 до 23). 13

`x.minute` - минуты (целое число от 0 до 59). 4

Объекты класса `timedelta` представляют разницу между двумя датами или значениями времени.

Справка:

```
>>> dir(timedelta)
```

```
['__abs__', '__add__', '__class__', '__delattr__', '__div__', '__doc__', '__eq__', '__floordiv__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '__mul__', '__ne__', '__neg__', '__new__',
'__nonzero__', '__pos__', '__radd__', '__rdiv__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rmul__',
'__rsub__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', 'days', 'max', 'microseconds', 'min',
'resolution', 'seconds', 'total_seconds']
```

Объекты класса `timedelta` обычно создаются при вычислении разности между двумя экземплярами класса `datetime` с помощью оператора вычитания (-). Однако имеется возможность создавать их и вручную, с помощью следующего конструктора класса:

```
datetime.timedelta ([days [, seconds [, microseconds [, milliseconds [, minutes [, hours [, weeks]]]]]]])
```

```
>>> print datetime.timedelta(hours=40)
1 day, 16:00:00
>>> print datetime.timedelta(minutes=40, seconds=25)
0:40:25
>>> t1 = datetime.datetime(1999, 4, 25)
>>> t2 = datetime.datetime(2015, 3, 14)
>>> td = t2 - t1
>>> td
datetime.timedelta(5802)
```

Полное описание директив `strftime` : <http://strftime.org/>

Определить дату, которая наступит через три дня; результат вывести в виде dd-mm-YYYY:

```
>>> from datetime import datetime
>>> from datetime import timedelta
>>> now = datetime.now()
>>> now
datetime.datetime(2015, 9, 2, 16, 44, 56, 290114)
>>> date_needed = now + timedelta(days=3)
>>> date_needed
datetime.datetime(2015, 9, 5, 16, 44, 56, 290114)
>>> date_formatted = date_needed.strftime("%d.%m.%Y")
>>> date_formatted
'05.09.2015'
>>> type(date_formatted)
<type 'str'>
```

Видим, что в итоге получаем строку.

ПРЕОБРАЗОВАНИЯ

При помощи `strftime` получаем данные строкового типа.

Если дана строка, которую необходимо преобразовать к типу дата-время, используем функцию **`strptime`** :


```
>>> date_string = '05.09.2015'
>>> date_datetime = datetime.strptime(date_string, "%d.%m.%Y")
>>> date_datetime
datetime.datetime(2015, 9, 5, 0, 0)
```

Если исходная строка содержит делители, отличные от указанных в формате, - возникнет ошибка:

```
>>> date_string = '05.09.2015'
>>> date_datetime = datetime.strptime(date_string, "%d-%m-%Y")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python2.7/_strptime.py", line 325, in _strptime
    (data_string, format))
ValueError: time data '05.09.2015' does not match format '%d-%m-%Y'
```

Для корректной работы функции `strptime`, нужно реализовать предварительные преобразования — привести в соответствие делители строки к делителям, указанным в желаемом формате даты:

```
>>> date_string = '05.09.2015'
>>> date_string_changed = date_string.replace('.', '-') # преобразования
>>> date_datetime = datetime.strptime(date_string_changed, "%d-%m-%Y")
>>> date_datetime
datetime.datetime(2015, 9, 5, 0, 0)
```

ПОЛЕЗНЫЕ ССЫЛКИ

- Полная документация к модулю `datetime` доступна на сайте python <https://docs.python.org/2/library/datetime.html>
- В частности, о функциях `strptime` и `strftime`: <https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior>
- Полное описание директив `strftime` (форматирование даты): <http://strftime.org/>
- Очень краткое введение в `datetime` <http://buddylindsey.com/python-date-and-datetime-objects-getting-to-know-them/>
- Больше примеров работы с датами: http://pleac.sourceforge.net/pleac_python/datesandtimes.html
- Об абсолютном (aware) и относительном (naive) времени: <http://asvetlov.blogspot.com/2011/02/date-and-time.html>