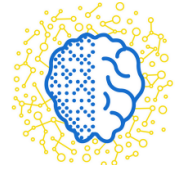


# Школа глубокого обучения

(<https://mipt.ru/science/labs/laboratoriya-neyronnykh-sistem-i-glubokogo-obucheniya/>)



**Физтех-Школа Прикладной математики и информатики МФТИ**

**Лаборатория нейронных сетей и глубокого обучения (DeerHackLab)**

## Занятие 1. Python и Jupyter Notebook

Захаркин Илья (ФИБТ 3 курс)



**Приветствуем Вас в Школе Глубокого Обучения от Физтех-Школы ПМИ МФТИ!**

Перед тем как начать складывать уровни нейросеточек как блинчики, нужно быть хорошо знакомым с библиотеками глубокого обучения и языком, на котором они написаны. В нашем курсе мы будем использовать язык **Python**, так как он является оптимальным сочетанием простоты, силы и количества библиотек глубокого обучения, написанных для него.

На этом занятии мы научимся:

- Установить интерактивную среду разработки на Python - Jupyter Notebook и разобраться, как в ней работать
- Установить сам Python
- Научиться (или освежить память) писать программы на Python, изучив его основы
- Научиться писать производительный код с помощью библиотеки NumPy
- Научиться строить графики с помощью библиотеки Matplotlib

## Установка Jupyter

Инструкция по установке Jupyter Notebook: <http://jupyter.readthedocs.org/en/latest/install.html> (<http://jupyter.readthedocs.org/en/latest/install.html>)

После установки запуск осуществляется командой в консоли:

```
jupyter notebook
```

Через несколько секунд должна открыться страница в браузере со списком файлов директории, в которой была запущена команда (можно не ждать и самим перейти по <http://localhost:8888/> (<http://localhost:8888/>) - обычно там располагается локальный "сервер" Jupyter Notebook`a).

## Онлайн Jupyter Notebook

Можно ничего не устанавливать и работать с сайтом <https://try.jupyter.org/> (<https://try.jupyter.org/>)

На сайте можно создать новый файл (New -> Python 2/3) или загрузить файл с компьютера с помощью Upload .

Будьте внимательны! **Сайт не сохраняет ваши файлы** и удаляет их после закрытия страницы. Чтобы загрузить файл себе откройте его нажмите File -> Download as -> ipynb .




Можно так же пользоваться сайтом <https://cloud.sagemath.com/> (<https://cloud.sagemath.com/>).

## Работа с Python в Jupyter Notebook

Чтобы создать новый файл, кликните New -> Python.

Ноутбук состоит из ячеек (cells), которые бывают текстовыми (Markdown) и кодовыми (Code). Выбрать тип ячейки можно на панели управления.

Работа с ячейками:

- Выбор ячейки — нажмите на нее мышкой.
- Редактирование — нажмите на нее два раза.
- Запуск ячейки — SHIFT+ENTER или нажмите на кнопку  на панели.
- Добавление новой ячейки — нажмите на кнопку  на панели.
- Удаление ячейки — нажмите на кнопку  на панели.
- Перемещение ячейки — нажмите на вертикальные стрелки.

Текстовые ячейки содержат в себе обычный текст, который может включать в себя формулы  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  и html -команды:

$$f(x) = \frac{1}{x}$$

$\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  cheat sheet - <https://wch.github.io/latexsheet/> (<https://wch.github.io/latexsheet/>)

## Основы Python

Сейчас существуют две часто используемые версии Питона — **Python 2** и **Python 3**. Эти версии довольно похожи, но есть отличия, из-за которых они **не являются совместимыми** - программы, написанные на одной версии языка, могут не работать в другой.

В нашем курсе мы будем писать на **Python 3**. Точная версия не принципиальна, но она должна быть  $\geq 3.5$

Если Вы пользуетесь каким-либо из дистрибутивов Linux, то Python скорее всего уже установлен. Попробуйте в терминале следующие команды для запуска интерактивного режима работы:

```
python или python3 или python2
```

Выход: Ctrl+D

Режим работы, в котором выполнится код из файла main.py

```
python main.py
```

Помощь: **help(X)** , где X — то, по чему нужна помощь.

Выход из помощи: q .

## Общая информация о языке

**Название** - «Питон» или «Пайтон» (в честь комедийных серий ВВС «Летающий цирк Монти-Пайтона»)

**Создатель** - голландец Гвидо ван Россум (**Guido van Rossum**) (в 1991 году)

**Особенности:**

- интерпретируемый
- объектно-ориентированный
- высокоуровневый язык
- встроенные высокоуровневые структуры данных
- динамическая типизация
- синтаксис прост в изучении
- поддержка модулей и пакетов (большинство библиотек бесплатны)
- универсальный
- интеграция с другими языками (C (Cython), C++, Java (JPython))

**Стиль оформления кода** - **PEP8** (если Вы хороший человек).

*Самое главное из PEP8:*

- отступ - 4 пробела
- длина строки < 80 символов
- переменные: var\_recommended
- константы: CONST\_RECOMMENDED

```
In [ ]: import this
```

## Типы

**Все типы данных** в Python относятся к одной из **2-х категорий: изменяемые (mutable) и неизменяемые (immutable)**.

*Неизменяемые объекты:*

- числовые данные (int, float),
- bool,
- None,
- символьные строки (class 'str'),
- кортежи (tuple).

*Изменяемые объекты:*

- списки (list),
- множества (set),
- словари (dict).

Вновь определяемые пользователем типы (классы) могут быть определены как неизменяемые или изменяемые. Изменяемость объектов определённого типа является принципиально важной характеристикой, определяющей, может ли объект такого типа **выступать в качестве ключа для словарей (dict)** или нет.

## int

```
In [ ]: x = 5
        print(x, '|', type(x))
```

```
In [ ]: a = 4 + 5
        b = 4 * 5
        c = 5 // 4
        print(a, b, c)
```

```
In [ ]: print( -(5 // 4) )
```

```
In [ ]: print( -5 // 4 )
```

```
In [ ]: x = 5 * 1000000000 * 1000000000 * 10**9 + 1
        print(x, '|', type(x))
```

## float

```
In [ ]: y = 12.345
        print(y, type(y))
```

```
In [ ]: a = 4.2 + 5.1
        b = 4.2 * 5.1
        c = 5.0 / 4.0
        print(a, b, c)
```

```
In [ ]: a = 5
        b = 4
        print(float(a) / float(b))
```

```
In [ ]: print(a / b)
```

## bool

```
In [ ]: a = True
        b = False

        print(a, '|', type(a))
        print(b, '|', type(b))
```

```
In [ ]: print(a + b)
        print(a + a)
        print(b + b)
```

```
In [ ]: print(int(a), int(b))
```

```
In [ ]: print(True and False, '\n')
        print(True or True, '\n')
        print(not False, '\n')
```

## None

```
In [ ]: z = None
        print(z, '|', type(z))
```

```
In [ ]: int(z)
```

```
In [ ]: if z is None:
        z = 'I am None!'
        z
```

## str

В *python2.7* есть отдельный тип **unicode**. В *python3.5* (и выше) (который будем использовать мы) всё это включено в тип **str**.

```
In [ ]: x = "abc"
        y = 'xyz'
        print(x, '|', type(x))
        print(y, '|', type(y))
```

```
In [ ]: a = 'Андрей'
        b = "Михайлович"
        s = a + " " + b
        print(s)
```

```
In [ ]: print(a.upper())
        print(a.lower())
```

```
In [ ]: print(len(a))
```

```
In [ ]: print(bool(a))
        print(bool("" + ''))
```

```
In [ ]: print(a)
        print(a[0])
        print(a[1])
        print(a[0:3])
```

```
In [ ]: print(a[0:4:2])
```

```
In [ ]: x = 'Роберт Дауни Младший'
print(x, type(x))
y = x.encode('utf-8')
print(y, type(y))
z = y.decode('utf-8')
print(z, type(z))
q = y.decode('cp1251')
print(q, type(q))
```

### Метод split():

```
In [ ]: splitted_line = "Райгородский Андрей Михайлович".split(' ')
print(splitted_line)
```

### tuple

```
In [ ]: t = ('a', 5, 12.345)
t
```

```
In [ ]: t.append(5)
```

```
In [ ]: dir(t)
```

```
In [ ]: t.index('a')
```

```
In [ ]: t[2]
```

```
In [ ]: m = (1, 2, 3)
t + m
```

```
In [ ]: t - m
```

```
In [ ]: len(m)
```

### Поменять переменные местами

```
In [ ]: a = -5
b = 100

a, b = b, a

print('a:', a, '\nb:', b)
```

### Немного про встроенные функции

В Python есть так называемые **магические** (или служебные) методы - они начинаются и заканчиваются на двойное нижнее подчёркивание: `__len__`, `__add__` ...

```
In [ ]: builtin
```

Также в Python есть **встроенные** функции и методы, которые являются в некотором смысле универсальными.

Функция `dir()` выводит список всех атрибутов (имён - методы, функции, классы..), которые есть у модуля/класса/объекта:

Давайте посмотрим на встроенные и служебные имена Python (не все):

```
In [ ]: dir( builtin )
```

## Задание 1

Выведите список атрибутов класса Exception.

```
In [6]: # Ваш код здесь
print(dir(Exception))

['_cause_', '__class__', '__context__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__sizeof__', '__str__', '__subclasshook__', '__suppress_context__', '__traceback__', 'args', 'with_traceback']
```

## Полезно

- При вызове метода какого-то класса (или функции какого-то модуля) можно написать его имя и через точку нажать **tab**:

`<имя_объекта_класса(модуля)>.[tab]`

Тогда всплывёт меню, в котором можно выбрать из всех существующих в этом классе методов (функций модуля).

```
In [ ]: builtin .help()
```

- Получение быстрой справки (`help()`) для любого объекта Python:

```
In [ ]: ? builtin
```

## Структуры данных и встроенные функции

### list

```
In [ ]: a = list()
b = []

print(a == b)
```

```
In [ ]: my_list = ['string', 100, 5.678, None]
my_list
```

- `list(range(start, end[, step]))` - получить последовательность (список) целых чисел, начинающуюся со `start`, заканчивающуюся в `end-1` и шагом `step`

```
In [ ]: array = range(1, 10, 2)
        print(array, '|', type(array))
```

```
In [ ]: array = list(array)
        print(array, '|', type(array))
```

```
In [ ]: array[1]
```

```
In [ ]: array[-1]
```

```
In [ ]: for i in range(5):
        print(i)
```

- Перевернуть список:

```
In [ ]: array = array[::-1]
        array
```

- Срезы ( `slice` 's') - это объекты языка Python, позволяющие получить какую-то часть итерируемого объекта.  
Пример:

```
In [ ]: foo = list(range(10))
        foo
```

```
In [ ]: foo[:5]
```

```
In [ ]: foo[5:]
```

```
In [ ]: foo[2:5]
```

```
In [ ]: slice_2_5 = slice(2, 5)
        print(slice_2_5, '|', type(slice_2_5))
```

```
In [ ]: foo[slice_2_5]
```

- `S.join(iterable)` - возвращает строку, которая является конкатенацией строк из `iterable`. Разделитель между строками - строка `S`

```
In [ ]: str_array = ['a', 'b', 'c', 'd', 'e']
```

```
In [ ]: ' '.join(str_array)
```

```
In [ ]: '! and !'.join(str_array)
```

- Списки можно "склеивать":

```
In [ ]: a = [1, 2, 3]
        b = [4, 5, 6]
        print(a + b)
```



## Методы класса list

```
In [ ]: dir(list)
```

- `L.append(element)` - добавляет элемент `element` в список `L`

```
In [ ]: l = [4, 5, 1, 3, 2]
        l.append('BANG!')
        l
```

- `sorted(iterable, key)` - возвращает объект, являющийся отсортированной в соответствии с компаратором (по ключу) `key` версией объекта `iterable`. **НЕ изменяет начальный объект!**

```
In [ ]: print(sorted(l), '|', l)
```

```
In [ ]: l.pop()
```

```
In [ ]: print(sorted(l), '|', l)
```

```
In [2]: def cmp(string):
        return len(string)
```

```
In [ ]: names = ['Александр', 'Василий', 'Анастасия', 'Соня', 'Френк', 'Оля']
        sorted(names, key=cmp)
```

```
In [ ]: names = ['Александр', 'Василий', 'Анастасия', 'Соня', 'Френк', 'Оля']
        sorted(names, key=lambda x: len(x))
```

```
In [ ]: names
```

- `L.sort(key)` - сортирует лист `L` в соответствии с компаратором (по ключу) `key`. **Изменяет начальный объект!**

```
In [ ]: l = [1, 2, 3, 4, 5]
        l.append('BANG!')
        l
        l.sort()
```

```
In [ ]: l.pop()
```

```
In [ ]: l
```

```
In [ ]: l.sort(reverse=True)
        l
```

- `L.count(element)` - возвращает количество вхождений элемента `element` в список `L`

```
In [ ]: l.count(1)
```

```
In [ ]: l.count('padabum')
```

```
In [ ]: len(l)
```

`L.index(element)` - возвращает индекс элемента `element` в списке `L`, если он там присутствует, `None` иначе

```
In [ ]: l.index(3)
```

## Задание 2

1. Создайте два списка одинаковых размеров из одинаковых элементов:  
    `items = [какие-то элементы]`  
    `shuffled_items = [эти же элементы, но расставленные в другом порядке]`
2. Отсортируйте список `items` в соответствии с ключом так, чтобы получился список `shuffled_items`

```
In [3]: # Ваш код здесь
# Исправлено _после_ проверки!!!
items = ["potatoes", "tomatoes", "and... unicorns!"]*7
print(items)
shuffled_items = ["and... unicorns!"]*7 + ["potatoes", "tomatoes"]*7
items.sort(key=Lambda x: -(cmp(x)))

print(items)
print(shuffled items)

['potatoes', 'tomatoes', 'and... unicorns!', 'potatoes', 'tomatoes', 'and...
.. unicorns!', 'potatoes', 'tomatoes', 'and... unicorns!', 'potatoes', 'to
matoes', 'and... unicorns!', 'potatoes', 'tomatoes', 'and... unicorns!', '
potatoes', 'tomatoes', 'and... unicorns!', 'potatoes', 'tomatoes', 'and...
unicorns!']
['and... unicorns!', 'and... unicorns!', 'and... unicorns!', 'and... unico
rns!', 'and... unicorns!', 'and... unicorns!', 'and... unicorns!', 'potato
es', 'tomatoes', 'potatoes', 'tomatoes', 'potatoes', 'tomatoes', 'potatoes
', 'tomatoes', 'potatoes', 'tomatoes', 'potatoes', 'tomatoes', 'potatoes',
'tomatoes']
['and... unicorns!', 'and... unicorns!', 'and... unicorns!', 'and... unico
rns!', 'and... unicorns!', 'and... unicorns!', 'and... unicorns!', 'potato
es', 'tomatoes', 'potatoes', 'tomatoes', 'potatoes', 'tomatoes', 'potatoes
', 'tomatoes', 'potatoes', 'tomatoes', 'potatoes', 'tomatoes', 'potatoes',
'tomatoes']
```

## Циклы - for и while

```
In [ ]: models = ['decision tree', 'linear model', 'svm', 'ensemble']

for model in models:
    print(model)
```

```
In [ ]: x = 100

while x > 50:
    x -= 10
    print(x)
```

## enumerate, zip

```
In [ ]: first = 'a b c d e f g'.split(' ')
        second = '1 2 3 4 5 6 7'.split(' ')

        zip(first, second)

In [ ]: list(zip(first, second))

In [ ]: methods = dir(__builtin__)

        for num, method in enumerate(methods):
            if num % 5 == 0:
                print(num, method)

In [ ]: enum = enumerate(first)
        list(enum)

In [ ]: enum = enumerate(first)
        zip_style = zip(range(0, len(first)), first)

        print(list(enum) == list(zip_style))
```

## Задание 3

1. Создайте список `a`, состоящий из каких-то элементов.
2. Создайте список `b` такого же размера, как `a`, состоящий из каких-то элементов.
3. Выведите нумерованный список пар из элементов списков `a` и `b`.

```
In [9]: # Ваш код здесь
a = ["The Mountains of Madness", "Magic Mirror", "Cthulhu", "Magic Wand",
     , "Invisibility Cloak", "Resurrection Stone"]
b = ["Jonathan Strange", "Harry Potter", "Nathaniel Chanticleer", "Gandalf",
     , "Sabriel", "Rincewind"]

resl = list(enumerate(zip(a, b)))
for e in resl:
    print(e[0], "-", e[1][0], "|", e[1][1])

0 - The Mountains of Madness | Jonathan Strange
1 - Magic Mirror | Harry Potter
2 - Cthulhu | Nathaniel Chanticleer
3 - Magic Wand | Gandalf
4 - Invisibility Cloak | Sabriel
5 - Resurrection Stone | Rincewind
```

## list comprehensions

```
In [ ]: a = [x for x in range(1, 6)]
        a

In [ ]: def f(x):
        return x ** 2
```

```
In [ ]: b = [f(x) for x in range(1, 10)]
        c = [x ** 2 for x in range(1, 10)]
        print(b, '==', c)
```

```
In [ ]: [x if x in 'aeiou' else '*' for x in 'apple']
```

```
In [ ]: def foo(i):
        return i, i + 1

        l = []
        for i in range(3):
            for x in foo(i):
                l.append(str(x))

        l
```

```
In [58]: l = [str(x) for i in range(3) for x in foo(i)]
        l
```

```
Out[58]: ['0', '1', '1', '2', '2', '3']
```

## Задание 4

Выведите список из 100 чисел через запятую. **Циклами пользоваться нельзя.**

```
In [1]: # Ваш код здесь
        # хотелось извратиться...)
        # print("".join(map(lambda x: str(x[0])+x[1], list(enumerate(["", "]*100)))))

        # :) range и map
        # print("".join(map(lambda x: str(x)+", ", range(100)))[-2])

        # рекурсия
        # def prnm(pr=0):
        #     if pr == 98:
        #         return "99"
        #     return str(pr) + ", " + prnm(pr+1)

        # print(prnm())

        # не круто - они не в порядке возрастания и повторяются
        # print(("1, 2, 3, 4, "*25)[-2])

        # если имелась в виду "магия" генераторов - то вот она:
        l = [i for i in range(100)]
        print(l. str ()[1:-2])

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39
, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57,
58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76
, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94,
95, 96, 97, 98, 9
```

## functions, lambdas

```
In [ ]: def make_coffee(size, sugar_dose=3, **kwargs):
        if sugar_dose > 5:
            return 'Too much sugar! Be careful! :('
        else:
            return 'Done: cup of {0} ml size; amount of sugar = {1}'.format(size,
```

```
In [ ]: make_coffee(100)
```

```
In [ ]: make_coffee(200, 1)
```

```
In [ ]: make_coffee(100, 6)
```

```
In [ ]: make_coffee(120, 5, name='Ilya', gender='male') # kwargs
```

```
In [ ]: negation = lambda x: -x
a = 5
print(negation(a))
```

## map, reduce, filter

- `map(func, iterables)` - выполняет преобразование `func` над элементами `iterables` и возвращает **новый** iterable :

```
In [52]: words = dir(list)
```

```
In [53]: letter_counts = list(map(lambda x: len(x), words))
```

```
print(letter_counts)
print()
print(words)

[7, 9, 12, 11, 11, 7, 7, 6, 10, 6, 16, 11, 6, 8, 8, 8, 8, 17, 8, 6, 7, 6,
7, 6, 7, 10, 13, 8, 12, 8, 11, 11, 10, 7, 16, 6, 5, 4, 5, 6, 5, 6, 3, 6, 7
, 4]
```

```
['_add_', '__class__', '__contains__', '__delattr__', '__delitem__', '__
dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__
getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__
init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__
ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed
__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__
subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'i
insert', 'pop', 'remove', 'reverse', 'sort']
```

```
In [ ]: l1 = [1,2,3,4]
l2 = [11, 12, 13, 14, 15]
l3 = [101, 102, 103]

triple_sum = list(map(lambda x, y, z: x + y + z, l1, l2, l3))
print(triple_sum)
```

- `reduce(func, iterables)` - производит вычисление с элементами последовательности, результатом которого является **одно значение**:

```
In [61]: from functools import reduce
```

```
In [62]: sum_of_counts = reduce(lambda x, y: x + y, letter_counts)
print(sum_of_counts)
378
```

- `filter(predicate, iterable)` - оставляет только те элементы, для которых **верен** предикат (функция, возвращающая bool):

```
In [ ]: mixed = ['мак', 'просо', 'мак', 'мак', 'просо', 'мак', 'просо', 'просо', 'п']
only_mac = list(filter(lambda x: x == 'мак', mixed))
print(only_mac)
```

## Задание 5

1. Дан массив строк: `['agfkd.,f', 'Qksdf;sb&..', 'asdoo*', 'bgf...d', 're54()kj[]..']`
2. Создайте список, состоящий из количества точек в каждой строке. Выведите его
3. Создайте новый список, в котором будут только строки, в которых более 2-х точек. Выведите его

Циклами пользоваться нельзя.

```
In [69]: # Ваш код здесь
strs = ['agfkd.,f', 'Qksdf;sb&..', 'asdoo*', 'bgf...d', 're54()kj[]..']
points = [i.count(".") for i in strs]
print(points)
strs_improved = list(filter(lambda x: x.count(".") > 2, strs))
print(strs_improved)
[1, 2, 0, 3, 1]
['bgf...d']
```

## set

```
In [ ]: s = set()
s
```

```
In [ ]: dir(s)
```

```
In [ ]: s.add(1)
s.add('a')
s.add(None)
s.add('bullet')
print(s)
```

```
In [ ]: s1 = set(range(0, 10))
s2 = set(range(5, 15))
```

```
In [ ]: print(s1.difference(s2))
print()
print(s2.difference(s1))
```

```
In [ ]: s1.intersection(s2)
```

```
In [ ]: s1.union(s2)
```

```
In [ ]: print('s1: ', s1, '\ns2: ', s2)
```

## dict

```
In [ ]: d = {}  
dd = dict()  
  
print(d == dd, '|', type(d))
```

```
In [ ]: dir(dict)
```

```
In [ ]: d['a'] = 100  
d
```

```
In [ ]: d = dict(short='dict', long='dictionary')  
d
```

```
In [ ]: d = dict([(1, 1), (2, 4)])  
d
```

```
In [ ]: d = dict.fromkeys(['a', 'b'])  
d
```

```
In [ ]: d = dict.fromkeys(['a', 'b'], 100)  
d
```

## dict comprehensions

```
In [ ]: d = {a: a ** 2 for a in range(7)}  
d
```

!

Будьте осторожны, если ключа, по которому поступил запрос, нет в словаре, то выбросит исключение:

```
In [ ]: d = {1: 100, 2: 200, 3: 300}  
d['a']
```

Поэтому безопаснее использовать **get(key)**. Тогда, если нужно, можно проверить на **None**:

```
In [ ]: d.get(1)
```

```
In [ ]: d.get('a') == None
```

Самое часто используемое - получение ключей, получение значений и получение всего вместе:

```
In [ ]: print(d.keys(), '|', type(d.keys()))  
  
print(list(d.keys()))
```

```
In [ ]: print(d.values(), '|', type(d.values()))  
  
        print(list(d.values()))
```

```
In [ ]: print(d.items(), '|', type(d.items()))  
  
        print(list(d.items()))
```

## modules

**Модули** - это "библиотеки" Python. То есть это самостоятельные, объединённые технически и логически, именованные части Python кода

- О модулях необходимо знать только одно - как их импортировать:

```
In [ ]: import collections
```

- Импортировать только какой-то компонент из модуля:

```
In [14]: from collections import Counter
```

- Импортировать с другим именем (чаще всего используется для лаконичности кода):

```
In [15]: import collections as cool lib
```

```
In [16]: count = cool lib.Counter()
```

Жизненный пример:

```
In [17]: import numpy as np
```

## files

```
In [80]: path = './20 newsgroups/sci.space/62478'
```

```
In [ ]: file = open(path, mode='r')  
        print([line for line in file][0])  
        file.close()
```



Режим	Обозначение
'r'	Открытие на <b>чтение</b> (является значением по умолчанию)
'rb'	Открытие на <b>чтение</b> , в предположении, что будут считываться <b>байты</b>
'w'	Открытие на <b>запись</b> , содержимое файла удаляется. Если файла не существует, создается <b>новый</b>
'wb'	Открытие на <b>запись байтов</b> , содержимое файла удаляется. Если файла не существует, создается <b>новый</b>
'a'	Открытие на <b>дозапись</b> , информация добавляется <b>в конец файла</b>
'r+'	Открыть файл на <b>чтение И запись</b> . Если файла нет, <b>новый НЕ создаётся</b>
'a+'	Открыть файл на <b>чтение И запись в конец файла</b> . Если файла нет, <b>новый создаётся</b>
't'	Открытие файла <b>как текстового</b> (по умолчанию)

```
In [ ]: with open(path, mode='r') as test_file:
        for line in test_file:
            print(line)
```

## classes

```
In [ ]: class Human:
        def __init__(self, name='', age=None, deep_learning_specialist=False):
            self.name = name
            self.age = age
            self.deep_learning_specialist = deep_learning_specialist

        def set_age(self, age):
            self.age = age

        def get_age(self):
            return self.age

        def __str__(self):
            return 'Name: {}\nAge: {} \
                    \nIs deep learning specialist: {}'.format(self.name, self.age,
                                                                self.deep_learning_specialist)

        class DLSchoolStudent(Human):
            def __init__(self, name='', age=None, deep_learning_specialist=False):
                super().__init__(name, age, deep_learning_specialist)
                self.total_grade = None
                self.deep_learning_specialist = True

            def __str__(self):
                return super().__str__()
```

```
In [ ]: human = Human('Person', 17)
        print(human)
```

```
In [ ]: student = DLSchoolStudent('Good Person', 18)
        print(student)
```

## exceptions

```
In [ ]: dir( builtin )
```

```
In [ ]: raise KeyboardInterrupt()
```

```
In [ ]: my_dict = {1: 100, 2: 200}
        print(my_dict['NEW'])
```

```
In [ ]: try:
        my_dict = {1: 100, 2: 200}
        print(my_dict['NEW'])
    except KeyError:
        print('Caught KeyError!')
```

## Полезные библиотеки Python

### glob

```
In [28]: import glob
```

```
In [ ]: !dir
```

```
In [29]: glob.glob('./[0-9].*')
```

```
Out[29]: []
```

```
In [30]: glob.glob('*.png')
```

```
Out[30]: ['dice.png']
```

```
In [ ]: glob.glob('?.jpg')
```

```
In [ ]: glob.glob('./**/', recursive=True)
```

```
In [ ]: glob.glob('**/*.txt', recursive=True)
```

### tqdm

- Устанавливаем виджеты:

```
pip install ipywidgets
```

(или `conda install -c conda-forge ipywidgets`)

- Разрешаем их использование в Jupyter Notebook:

```
jupyter nbextension enable --py --sys-prefix widgetsnbextension
```

- Перезагружаем ядро (Restart Kernel)

- Устанавливаем tqdm:

```
pip install tqdm
```

(или `conda install -c conda-forge tqdm`)

Больше про tqdm: <https://pypi.python.org/pypi/tqdm> (<https://pypi.python.org/pypi/tqdm>)

```
In [18]: from tqdm import tqdm
        from tqdm import trange, tqdm_notebook
        from time import sleep
```

```
In [20]: cnt = 0
         for i in tqdm(range(1000)):
             sleep(0.01)
             cnt += 1
100%|██████████| 1000/1000 [00:10<00:00, 97.40it/s]
```

```
In [ ]: for i in trange(10, desc='1st loop'):
         for j in tqdm_notebook(range(100), desc='2nd loop', leave=False):
             sleep(0.01)
```

## **collections**

- `defaultdict()` - класс словаря, у которого есть значение по умолчанию - порой очень пригождается:

```
In [2]: from collections import defaultdict
```

```
In [23]: d = defaultdict(int)
         print(d['key'])
         d['key'] = 5
         print(d['key']) # 5
0
5
```

```
In [24]: d = defaultdict(lambda: 'empty')
         print(d['key'])
         d['key'] = 'full'
         print(d['key'])
empty
full
```

```
In [25]: d = defaultdict(list)
         print(d)
         d['list1'].append(100)
         d['list1'].append(200)
         print(d)
         print(d['list1'])
defaultdict(<class 'list'>, {})
defaultdict(<class 'list'>, {'list1': [100, 200]})
[100, 200]
```

- `Counter()` - класс словаря, предназначенного для счётчиков. По сути, == `defaultdict(int)` :

```
In [26]: from collections import Counter
```

In [27]: counter = Counter()

```
for word in dir(__builtin__):  
    for letter in word:  
        counter[letter] += 1 # или .update(value)
```

print(counter)

```
Counter({'r': 224, 'o': 136, 'e': 132, 't': 98, 'n': 98, 'i': 85, 'a': 63,  
's': 59, 'E': 57, 'c': 48, 'l': 43, 'd': 39, 'p': 38, '_': 38, 'm': 31, 'u':  
' ': 29, 'y': 22, 'g': 21, 'b': 16, 'I': 14, 'x': 13, 'h': 12, 'W': 11, 'f':  
9, 'F': 8, 'v': 8, 'N': 8, 'A': 7, 'P': 7, 'R': 7, 'S': 7, 'U': 7, 'O': 6,  
'D': 6, 'T': 6, 'B': 5, 'k': 5, 'C': 5, 'L': 3, 'w': 3, 'K': 2, 'M': 2, 'z':  
' ': 2, 'G': 1, 'V': 1, 'Z': 1, 'Y': 1, 'H': 1, 'j': 1})
```

## Задание 6

1. Создайте словарь счётчиков
2. Создайте переменную, в которую сохраните все пути к нужным текстовым файлам (*расположены по адресу ' ./20\_newsgroups/sci.space/'* )
3. Для каждого текста (текстового файла) посчитайте сколько каждое слово (из этого текста) встретилось в этом тексте (используйте предыдущие пункты)

```
In [3]: # Ваш код здесь
from collections import Counter, defaultdict
import glob

files = glob.glob("./20_newsgroups/sci.space/*")
# print(files)

dofc = defaultdict(Counter)

for f in files:
    dofc[f] = Counter()
    with open(f, mode="r") as fl:
        for line in fl:
            for w in line.split():
                dofc[f][w] += 1

# print(dofc)
for text in dofc:
    print("B", text, "есть слова: ")
    for word in dofc[text]:
        print("---", word, "в количестве", dofc[text][word])
    print()

B ./20_newsgroups/sci.space/62479 есть слова:
--- Newsgroups: в количестве 1
--- sci.space в количестве 2
--- Path: в количестве 1
--- cantaloupe.srv.cs.cmu.edu!das-news.harvard.edu!noc.near.net!howland.re
ston.ans.net!zaphod.mps.ohio-state.edu!cs.utexas.edu!uunet!nih-csl!NewsWat
cher!user в количестве 1
--- From: в количестве 1
--- england@helix.nih.gov в количестве 1
--- (Mad в количестве 1
--- Vlad) в количестве 1
--- Subject: в количестве 1
--- Satellite в количестве 1
--- Capabilities-Patriot в количестве 1
--- Games в количестве 1
--- Message-ID: в количестве 1
--- <england-170593093754@156.40.182.12> в количестве 1
--- Followup-To: в количестве 1
--- Sender: в количестве 1
--- postman@helix.nih.gov в количестве 1
```

То, что Вы реализовали выше, есть ни что иное, как простая реализация **bag-of-words (мешок слов)** для корпуса из 9 документов.

## Список материалов для самостоятельного изучения

- Сайт языка Python - <https://www.python.org/> (<https://www.python.org/>)
- Курс Python с нуля, можно выполнять задания в интерактивном режиме - <http://pythontutor.ru/> (<http://pythontutor.ru/>)

- *Новый онлайн-курс по Питону на Coursera от Mail.Ru Group* - <https://www.coursera.org/learn/programming-in-python> (<https://www.coursera.org/learn/programming-in-python>)
- *Самоучитель Python* - <https://pythonworld.ru/samouchitel-python> (<https://pythonworld.ru/samouchitel-python>)
- *Статья про коварности Python* - <https://habrahabr.ru/company/mailru/blog/337364/> (<https://habrahabr.ru/company/mailru/blog/337364/>)
- *Очень полезные трюки в Jupyter Notebook*: <https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/> (<https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/>)