

# Семинар 5

Введение в программирование на Python

Папулин С.Ю.  
papulin\_hse@mail.ru

2015

1. Строки
2. Списки
3. Функции
4. Модули
5. Пакеты

1. Словари
2. Множества
3. Работа с файлами
4. Matplotlib

# Словари

# Словари

**Словарь** (dictionary). Множество пар вида ключ-значение. Ключ должен быть уникальным.

```
d = { 4: "hello", "aa": 1, 2: [1, 2, 3], (1, 2, 3): "world" }
```

d[4] → 'hello'

d["aa"] → 1

d[2] → [1, 2, 3]

d[(1, 2, 3)] → 'world'

# Создание словаря

1 `d = {1:"value1", 2:"value2", 3:"value3"}`

2 `d = {}`  
`d[1] = "value1"`  
`d[2] = "value2"`  
`d[3] = "value3"`

3 `d = dict([(1, "value1"), (2, "value2"), (3, "value3")])`

4 Если ключи являются строками, то

```
d = dict(key1="value1", key2="value2", key3="value3")  
  
{'key3': 'value3', 'key1': 'value1', 'key2': 'value2'}
```

# Функция len и оператор IN

len(dict)	Количество элементов в словаре dict	len({1:1,2:1,3:2})=3
-----------	-------------------------------------	----------------------

## Ключевое слово IN и NOT IN

```
myD={ 3:5, 5:22, 7:99}
```

```
3 in myD
```

True

```
3 not in myD
```

False

# Добавление пары, возвращение значения и метод get

Добавление нового элемента словаря

```
d = {1: "value1", 2: "value2", 3: "value3"}
```

```
d[5] = "value5"
```



```
{1: 'value1', 2: 'value2', 3: 'value3', 5: 'value5'}
```

Значение по ключу

```
d[5] → 'value5'
```

Метод **get**(*key*): если элемент с ключом *key* присутствует в словаре, то возвращает его значение, если нет - None

```
d.get(2) → 'value2'
```



# Методы items

`dict.items()` возвращает итеративный объект view пар словаря

```
d = {1:"value1", 2:"value2", 3:"value3"}
```

```
d.items()
```

```
dict_items([(1, 'value1'), (2, 'value2'), (3, 'value3')])
```

```
t = [(k, v) for k, v in d.items()]
```



```
[(1, 'value1'), (2, 'value2'), (3, 'value3')]
```

## Методы values и keys

`dict.values()` возвращает итеративный объект view значений словаря

```
d = {1:"value1", 2:"value2", 3:"value3"}
```

`d.values()`  `dict_values(['value1', 'value2', 'value3'])`

`[v for v in d.values()]`  `['value1', 'value2', 'value3']`

`dict.keys()` возвращает итеративный объект view ключей словаря

`d.keys()`  `dict_keys([1, 2, 3])`

`[v for v in d.keys()]`  `[1, 2, 3]`

# Метод copy

`dict.copy()` возвращает копию словаря

```
d1 = {1: 'value1', 2: 'value2', 3: 'value3'}  
d2 = d1.copy()  
d2[2] = "changed"  
print("d1 -> " + str(d1))  
print("d2 -> " + str(d2))
```



```
d1 -> {1: 'value1', 2: 'value2', 3: 'value3'}  
d2 -> {1: 'value1', 2: 'changed', 3: 'value3'}
```

# Метод update

`dict.update(dict2)` расширяет словарь `dict` за счет элементов словаря *dict2*

```
d = {1: "value1", 2: "value2", 3: "value3"}
```

```
d2 = {4: "value4", 5: "value5"}
```

```
d.update(d2)
```



```
{1: 'value1', 2: 'value2', 3: 'value3', 4: 'value4', 5: 'value5'}
```

# Метод clear и оператор del

`dict.clear()` – очищает словарь `dict`

```
d = {1: "value1", 2: "value2", 3: "value3"}
```

```
d.clear() → {}
```

`del dict[key]` – удаляет элемент словаря `dict` с ключом *key*

```
d = {1: "value1", 2: "value2", 3: "value3"}
```

```
del d[2] → {1: 'value1', 3: 'value3'}
```

# Сортировка по ключу

```
d = {2:"f", 1:"tt", -5:"aaaa", 4:"ppp"}
```

*#Сортировка по ключу*

1 lSortedByVal = sorted(d.items())

2 lSortedByVal = sorted(d.items(), reverse=True)

1 [(-5, 'aaaa'), (1, 'tt'), (2, 'f'), (4, 'ppp')]

2 [(4, 'ppp'), (2, 'f'), (1, 'tt'), (-5, 'aaaa')]

3 lSortedByVal = sorted(d.items(), key=keySortByKey, reverse=False)

4 lSortedByVal = sorted(d.items(), key=lambda x:x[0], reverse=False)

3 [(-5, 'aaaa'), (1, 'tt'), (2, 'f'), (4, 'ppp')]

4 [(-5, 'aaaa'), (1, 'tt'), (2, 'f'), (4, 'ppp')]

```
def keySortByKey(x):  
    return x[0]
```

# Сортировка по значению

```
d = {2:"f", 1:"tt", -5:"aaaa", 4:"ppp"}
```

*#Сортировка по значению*

1 lSortedByVal = sorted(d.items(), key=keySortByValue, reverse=False)

2 lSortedByVal = sorted(d.items(), key=lambda x: x[1], reverse=False)

1 [(-5, 'aaaa'), (2, 'f'), (4, 'ppp'), (1, 'tt')]

2 [(-5, 'aaaa'), (2, 'f'), (4, 'ppp'), (1, 'tt')]

```
def keySortByValue(x):  
    return x[1]
```

*#Сортировка по числу символов значений словаря*

3 lSortedByVal = sorted(d.items(), key=keySortByLenValue,  
reverse=False)

4 lSortedByVal = sorted(d.items(), key=lambda x: len(x[1]),  
reverse=False)

3 [(2, 'f'), (1, 'tt'), (4, 'ppp'), (-5, 'aaaa')]

4 [(2, 'f'), (1, 'tt'), (4, 'ppp'), (-5, 'aaaa')]

```
def keySortByLenValue(x):  
    return len(x[1])
```

# Сортировка ключей словаря

```
d = {2:"f", 1:"tt", -5:"aaaa", 4:"ppp"}
```

*#Сортировка ключей*

- 1 lSortedKey = sorted(d.keys())
- 2 lSortedKey = sorted(d)
- 3 lSortedKey = sorted(d, reverse=True)

1 [-5, 1, 2, 4]      2 [-5, 1, 2, 4]

3 [4, 2, 1, -5]

*#Сортировка ключей по модулю*

- 4 lSortedKey = sorted(d, key=lambda x:abs(x))
- 5 lSortedKey = sorted(d, key=keySortABS)

4 [1, 2, 4, -5]

5 [1, 2, 4, -5]

```
def keySortABS(x):  
    return abs(x)
```



# Сортировка значений словаря

```
d = {2:"f", 1:"tt", -5:"aaaa", 4:"ppp"}
```

*#Сортировка значений*

1 lSortedKey = sorted(d.values())

2 lSortedKey = sorted(d.values(), reverse=True)

1 ['aaaa', 'f', 'ppp', 'tt']

2 ['tt', 'ppp', 'f', 'aaaa']

*#Сортировка значений словаря: ключи*

3 lSortedKey = sorted(d, key=d.get) 3 [-5, 2, 4, 1]

*#Сортировка значений по количеству символов*

4 lSortedKey = sorted(d.values(), key=lambda x:len(x))

5 lSortedKey = sorted(d.values(), key=keySortLenVal)

4 ['f', 'tt', 'ppp', 'aaaa']

5 ['f', 'tt', 'ppp', 'aaaa']

```
def keySortLenVal(x):  
    return len(x)
```

# FOR и словарь

```
d = {1:"value1", 2:"value2", 3:"value3"}
```

- 1 

```
list1=[]  
for k, v in d.items():  
    t=(k, v)  
    list1.append(t)  
print("list1 -> " + str(list1))
```
- 2 

```
list2 = [(k, v) for k, v in d.items()]  
print("list2 -> " + str(list2))
```

list1 -> [(1, 'value1'), (2, 'value2'), (3, 'value3')]

list2 -> [(1, 'value1'), (2, 'value2'), (3, 'value3')]

## FOR и словарь

```
d = {1:"value1", 2:"value2", 3:"value3"}
```

```
d1 = {k:v for k, v in d.items() if k>2}
```



```
{3: 'value3'}
```

Количество одинаковых элементов в списке?

```
lNumbers = ["a", "r", "v", "s", "n", "a", "v", "a", "b"]
d = {}
for el in lNumbers:
    if (el in d):
        d[el] += 1
    else:
        d[el] = 1
print(d)
```



```
{'r': 1, 's': 1, 'v': 2, 'b': 1, 'n': 1, 'a': 3}
```

# Множества

# Множества

**Множество** (set) – неупорядоченная коллекция без повторяющихся элементов

$$s = \{3, \text{"e12"}, (1, 2), 8\}$$

- Уникальные элементы

$$s = \{3, \text{"e12"}, (1, 2), 8, 3\} \longrightarrow \{8, (1, 2), \text{'e12'}, 3\}$$

- В качестве элементов можно использовать только кэшируемые объекты. Например, списки нельзя, кортежи можно
- Не поддерживается индексация

# Создание множества

## 1 Фигурные скобки

```
s = {3, "e12", (1, 2), 8}
```

## 2 Функция **set()**

```
l = [3, "e12", (1, 2), 8]
```

```
s = set(l) → {8, (1, 2), 'e12', 3}
```

```
st = "hello"
```

```
s = set(st) → {'l', 'e', 'o', 'h'}
```

Пустое множество ~~{ }~~

```
s = set()
```

# Операции

$a = \{1, 2, 3, 4\}$

$b = \{3, 4, 5, 6\}$

## 1 Объединение ( $|$ )

$a \mid b \longrightarrow \{1, 2, 3, 4, 5, 6\}$

## 2 Пересечение ( $\&$ )

$a \& b \longrightarrow \{3, 4\}$

## 3 Разность ( $-$ )

$a - b \longrightarrow \{1, 2\}$

## 4 Симметрическая разность ( $\wedge$ )

$a \wedge b \longrightarrow \{1, 2, 5, 6\}$



## Методы add и remove

**set.add(*el*)** – добавляет элемент *el* в множество set

a = {1, 2, 3, 4}

a.add(5)  {1, 2, 3, 4, 5}

**set.remove(*el*)** – удаляет элемент *el* из множества set

a = {1, 2, 3, 4}

a.remove(1)  {2, 3, 4}

# Методы copy и clear

`set.copy()` возвращает копию множества

```
a = {1, 2, 3, 4}
```

```
a1 = a.copy() → {1, 2, 3, 4}
```

`set.clear()` – очищает множество set


```
a = {1, 2, 3, 4}
```

```
a.clear() → []
```

Объект списка

## Функция sorted для множества

`sorted(set, key, reverse)` возвращает отсортированный список элементов множества *set* по функции *key* в порядке *reverse*

`s = {1, 4, 5, -4}`  `{1, -4, 5, 4}`

`sorted(s)`  `[-4, 1, 4, 5]`

`sorted(s, reverse = True)`  `[5, 4, 1, -4]`

`sorted(s, key=lambda x:abs(x), reverse = True)`



`[5, -4, 4, 1]`

# Set vs Frozenset

**set**(*list*) создает изменяемое множество уникальных элементов из списка *list*

**frozenset**(*list*) создает неизменяемое множество уникальных элементов из списка *list*

Формируется новое множество colors2 из элементов colors1, в которых нет буквы "r"

```
colors1 = {"red", "green", "blue", "yellow"}  
colors2 = {el for el in colors1 if "r" not in el}  
print(colors2)
```



{'blue', 'yellow'}

# Работа с файлами

**Файловый объект** (file object) обеспечивает доступ к ресурсам на диске или других типах хранения и коммуникационных устройств (input/output, буферы памяти, сокеты и пр.)

Файловый объект также называют файл подобные объекты или потоки (streams)

Существует **три категории** файловых объектов:

- Исходные бинарные файлы
- Буферизированные бинарные файлы
- Текстовые файлы

# Создание и закрытие файлового объекта

`fObj = open(filename, mode)` – возвращает файловый объект

Имя файла

Режим доступа

‘r’ – чтение (по умолчанию, если опущен аргумент *mode*)

‘w’ – запись (существующих файл будет перезаписан)

‘a’ – открывает файл для добавления в конец

‘r+’ – чтение и запись

‘b’ – бинарный режим

‘t’ – текстовый режим

После завершения работы с файловым объектом, необходимо его закрыть посредством метода **close()**

`fObj.close()`



# Чтение строк из файла. Метод readline

`fObj.readline()` – возвращает строку символов. При чтении строки курсор смещается

```
fObj = open("input.txt", "rt")  
print(repr(fObj.readline()))  
print(repr(fObj.readline()))
```



Hello, world!

I'm your program



'Hello, world!\n'

"I'm your program"

```
fObj = open("input.txt", "rt")  
print(fObj.readline().rstrip())  
print(fObj.readline().rstrip())
```



Hello, world!

I'm your program

# Чтение строк из файла. Метод read

`fObj.read(num)` – возвращает последовательно *num* символов. При чтении символа курсор смещается

Input.txt

```
Hello, world!  
I'm your program
```

```
fObj = open("input.txt", "rt")  
print(fObj.read(2))  
print(fObj.read(2))  
print(fObj.read(2))
```



```
He  
ll  
o,
```

# Чтение строк из файла с FOR

```
fObj = open("input.txt", "rt")  
for line in fObj:  
    print(line.rstrip())
```



Hello, world!  
I'm your program

Input.txt

Hello, world!  
I'm your program

# Чтение строк из файла. Метод seek и tell

`fObj.seek(num)` – смещает курсор на *num* символов

```
fObj = open("input.txt", "rt")  
fObj.seek(4)  
print(fObj.read())
```



o, world!  
I'm your program

Input.txt

```
Hello, world!  
I'm your program
```

`fObj.tell()` – возвращает текущую позицию курсора

## Запись в файл

`fObj.write(content)` – заменяет содержание файла на *content*, устанавливает курсор в конце файла и возвращает количество записанных элементов

```
fObj = open("input.txt", "wt")  
line = "This is a new line.\n"  
fObj.write(line)  
fObj.write(line)
```

Input.txt

```
Hello, world!  
I'm your program
```



Input.txt

```
This is a new line.  
This is a new line.
```

# Оператор WITH

Использование **WITH** позволяет гарантированно закрыть файловый объект без явного применения метода **close()**

```
with open ("input.txt", "rt") as fObj :  
    print (fObj.read() )
```



Hello, world!  
I'm your program

# Enumerate для файлового объекта

```
fObj = open("input.txt", "r")  
for indx, line in enumerate(fObj):  
    print("Line " + str(indx) + ": " + line.rstrip())
```



Line 0: Hello, world!

Line 1: I'm your program

[Программирование на Python. Часть 3: Списки в питоне](#)

[Программирование на Python: Часть 4. Словари](#)

[5. Data Structures](#)

[7. Input and Output](#)

[Dictionaries](#)

[Sorting HOW TO](#)

[sort dict by value python](#)