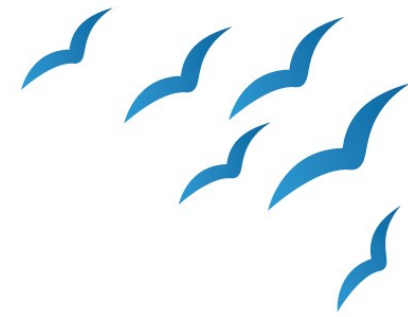


Коллекции





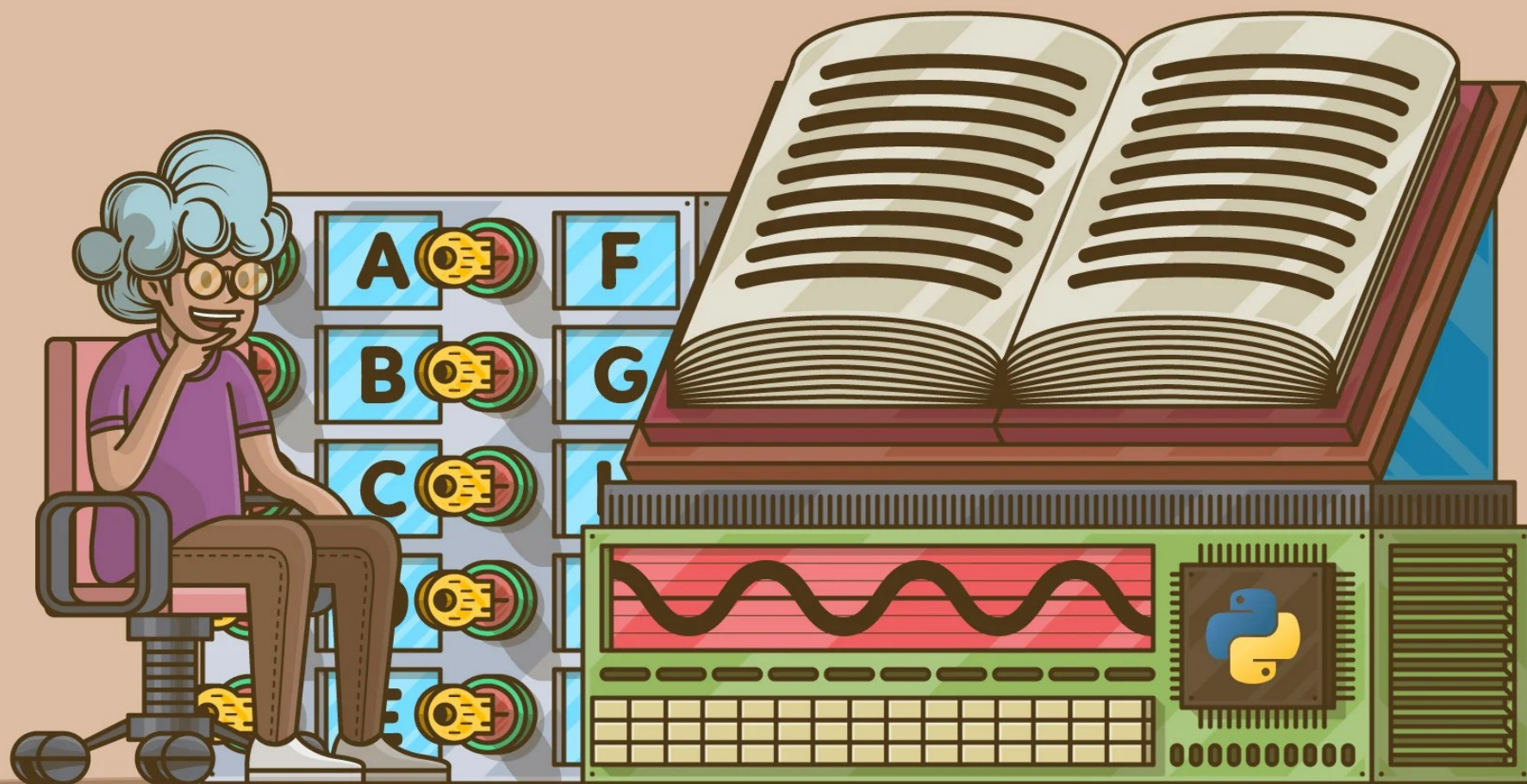
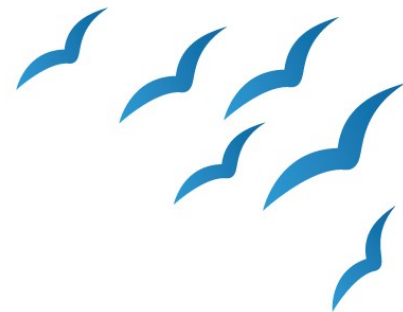
Оглавление

1. Строка (`str`)
2. Список (`list`)
3. Словарь (`dict`)
4. Множество (`set`)
5. Замороженное множество (`frozenset`)
6. Кортеж (`tuple`)

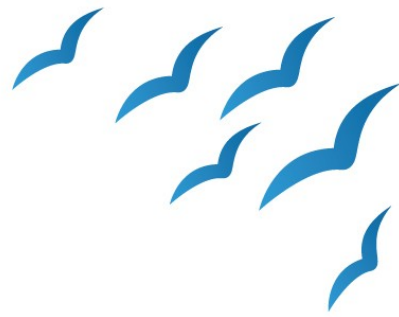


Тип коллекции	Изменяемость	Индексированность	Уникальность	Как создаём
Список (list)	+	+	-	<code>[]</code> <code>list()</code>
Кортеж (tuple)	-	+	-	<code>()</code> , <code>tuple()</code>
Строка (string)	-	+	-	<code>"</code> <code>" "</code>
Множество (set)	+	-	+	<code>{elm1, elm2}</code> <code>set()</code>
Неизменяемое множество (frozenset)	-	-	+	<code>frozenset()</code>
Словарь (dict)	+ элементы - ключи + значения	-	+ элементы + ключи - значения	<code>{}</code> <code>{key: value,}</code> <code>dict()</code>

Словарь



Real Python



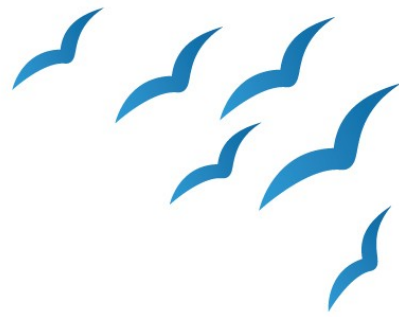
Определение словаря

```
dict = {k:v}
```

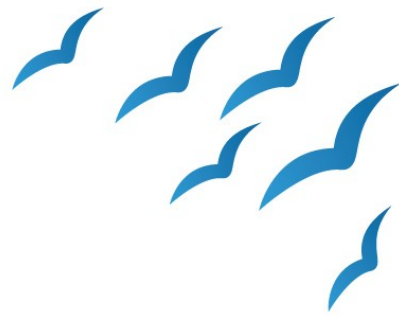
Словарь задается парой **ключ: значение**,

```
dic = {  
    <key>: <value>,  
    <key>: <value>,  
    .  
    .  
    .  
    <key>: <value>  
}
```

Пример 1:



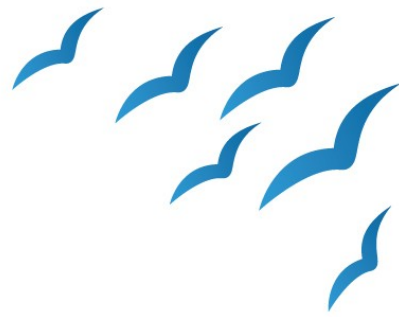
```
person = {  
    'name': 'Маша',  
    'login': 'masha',  
    'age': 25,  
    'email': 'masha@yandex.ru',  
    'password': 'fhei23jj~'  
}  
  
print (type (person) )  
<class 'dict'>
```



Пример 2:

#Словарь, где ключи являются
целыми числами.

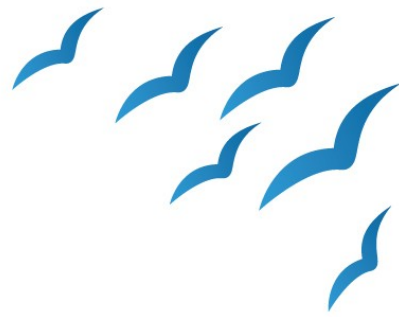
```
dict_sample = {  
    1: 'mango',  
    2: 'coco'  
}
```



Пример 3:

```
# Hmm... если ключи состоят из  
примитивных типов то могу ли я  
сделать так ?
```

```
dict_sample = {  
    True: 'mango',  
    False: 'coco'  
}
```

Пример 4:

```
# .. пойдём дальше
```

```
dict_sample = {  
    None: 'mango',  
    None: 'coco'  
}
```



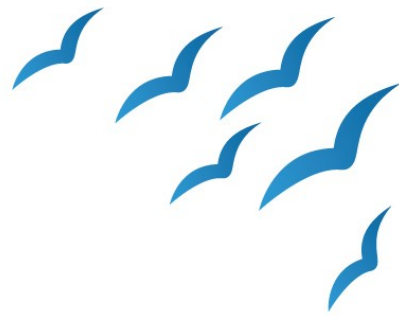
Другие способы создания

```
{ 'name': 'Маша', 'age': 16 }      # литеральным  
выражением
```

```
person = {}      # динамическое присваение по ключам  
person['name'] = 'Маша'  
person['age'] = 16
```

!!!

```
dict(name='Маша', age=16) # через конструктор dict
```



Доступ к элементу по ключу

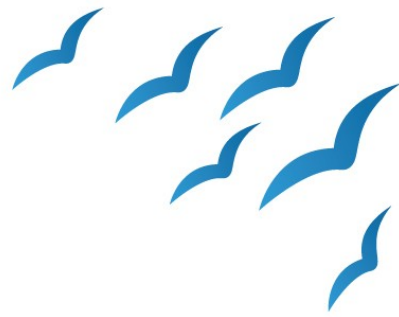
```
dict = { k: v }
```

```
>>> person['name']
```

Маша

Замена значения

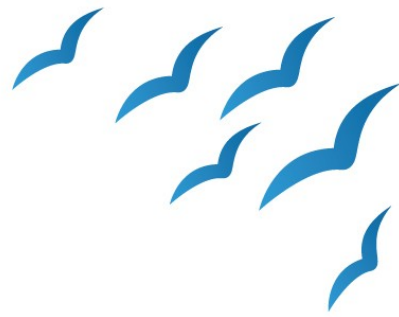
```
>>> person['name'] = 'Даша'
```



Добавление нового элемента

```
dict = { k: v, k2: v2 }
```

```
>>> person['surname'] = 'Медведьева'
{
    'name': 'Даша',
    'login': 'masha',
    'age': 25, 'email': 'masha@yandex.ru',
    'password': 'fhei23jj~',
    'surname': 'Медведьева'
}
```

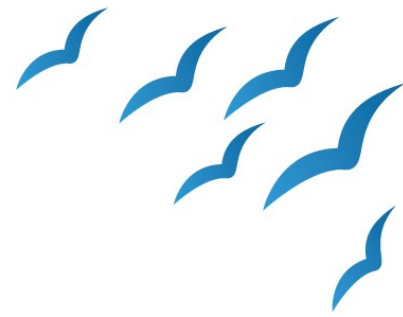


Удаление элемента

dict = { k: v, :  }

```
>>> del person['login']  
{  
    'name': 'Даша',  
    'age': 25,  
    'email': 'masha@yandex.ru',  
    'password': 'fhei23jj~',  
    'surname': 'Медведьева'  
}
```

Проверка на наличие ключа



```
dict = { "A": v }
```

"A"? ↗

```
>>> 'name' in person
```

```
True
```

```
print('Ключ есть') if ('name' in person)  
else print('Ключа нет')
```

Длина словаря в Python



```
dict = { k : v , k2 : v2 }
```

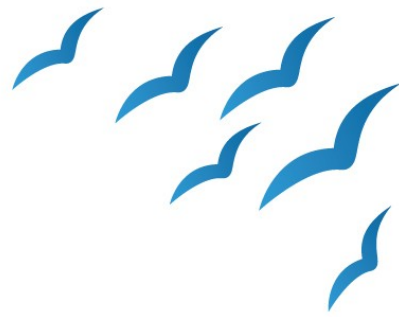
len = 2 ↗

Количество записей мы можем получить, воспользовавшись функцией len()

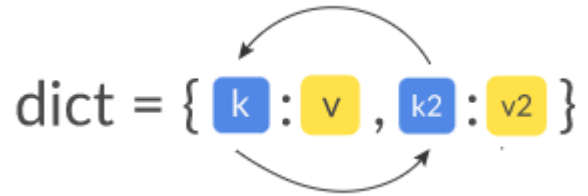
```
>>> num_of_items = len(person)
```

```
>>> print(num_of_items)
```

```
>>> 5
```



Сортировка словаря

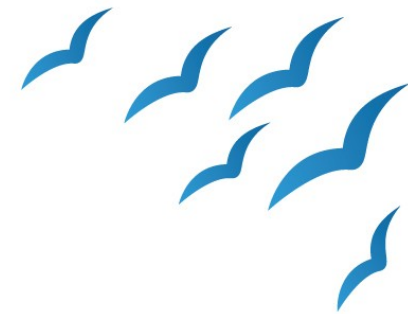


```
statistic_dict = {'b': 10, 'd': 30, 'e': 15,  
'c': 14, 'a': 33}
```

```
for key in sorted(statistic_dict):  
    print(key)
```

```
a  
b  
c  
d  
e
```


Итерирование словаря



dict = { k : v , k2 : v2 , k3 : v3 }



```
statistic_dict = {'b': 10, 'd': 30, 'e': 15,  
'c': 14, 'a': 33}
```

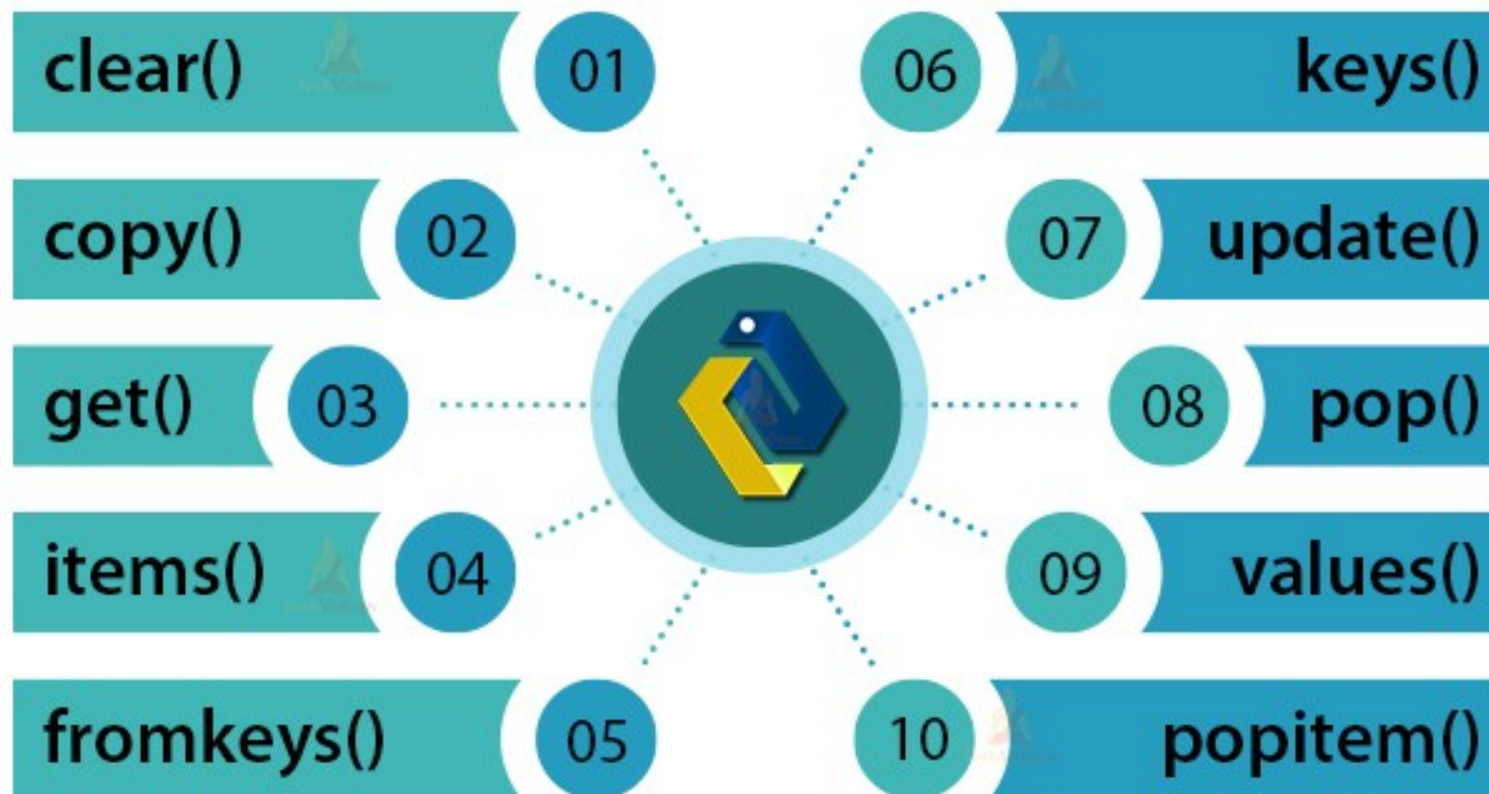
```
for key, val in statistic_dict.items():  
    print(key)  
    print(val)
```

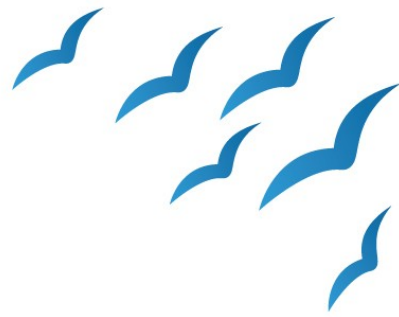
предположение

Догадка, предварительная мысль.



Python Dictionary Methods



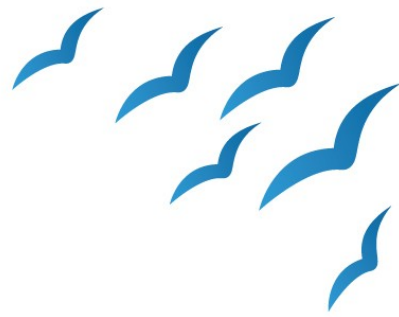


clear()

Метод производит удаление всех элементов из словаря.

```
>>> x = {'one': 0, 'two': 20, 'three': 3,
'four': 4}
>>> x.clear()
>>> x

# {}
```



copy()

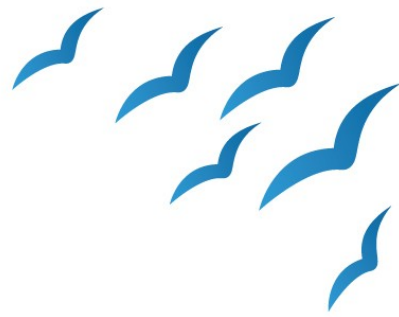
Метод создает копию словаря.

```
>>> x = {'one': 0, 'two': 20, 'three': 3,  
        'four': 4}
```

```
>>> y = x.copy()
```

```
>>> y
```

```
{'one': 0, 'two': 20, 'three': 3, 'four': 4}
```



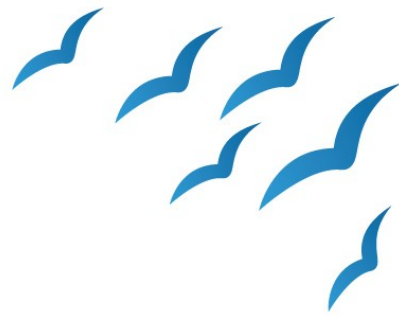
get(key[, default])

Метод dict.get() возвращает значение для ключа key, если ключ находится в словаре, если ключ отсутствует то вернет значение default.

Если значение default не задано и ключ key не найден, то метод вернет значение None.

Метод dict.get() никогда не вызывает исключение KeyError, как это происходит в операции получения значения словаря по ключу [dict[key]].

```
>>> x = {'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> x.get('two', 0)
# 2
>>> x.get('ten', 0)
# 0
```



items()

Метод `dict.items()` возвращает новый список кортежей вида `(key, value)`, состоящий из элементов словаря.

```
>>> x = {'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> items = x.items()
>>> items
```

```
dict_items([('one', 1), ('two', 2), ('three', 3),
('four', 4)])
```



fromkeys(iterable[, value])

Метод dict.fromkeys() встроенного класса dict() создает новый словарь с ключами из последовательности iterable и значениями, установленными в value.

```
>>> x = dict.fromkeys(['one', 'two', 'three',  
    'four'])
```

```
>>> x
```

```
{'one': None, 'two': None, 'three': None, 'four':  
None}
```

```
>>> x = dict.fromkeys(['one', 'two', 'three',  
    'four'], 0)
```

```
>>> x
```

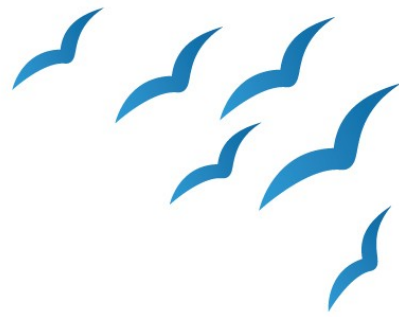
```
{'one': 0, 'two': 0, 'three': 0, 'four': 0}
```




keys()

Метод `dict.keys()` возвращает новый список-представление всех ключей , содержащихся в словаре `dict`.

```
>>> x = {'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> keys = x.keys()
>>> keys
dict_keys(['one', 'two', 'three', 'four'])
```



keys()

Список-представление ключей `dict_keys`, является динамичным объектом. Это значит, что все изменения, такие как удаление или добавление ключей в словаре сразу отражаются на этом представлении.

```
# Производим операции со словарем 'x', а все
# отражается на списке-представлении `keys`
>>> x = {'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> keys = x.keys()
>>> del x['one']
>>> keys
dict_keys(['two', 'three', 'four'])
>>> x
{'two': 2, 'three': 3, 'four': 4}
```

update() - объединение словарей


dict1 = { k1: v1 }
dict2 = { k2: v2 }) join

```
showcase_1 = { 'Apple': 2.7, 'Grape': 3.5,  
               'Banana': 4.4 }
```

```
showcase_2 = { 'Orange': 1.9, 'Coconut': 10 }  
showcase_1.update(showcase_2)
```

```
print(showcase_1)
```

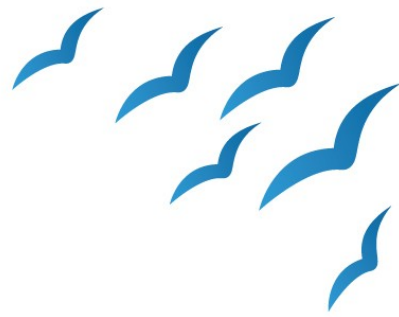
```
> { 'Apple': 2.7, 'Grape': 3.5, 'Banana': 4.4,  
    'Orange': 1.9, 'Coconut': 10 }
```



pop(key[, default])

Метод dict.pop() вернет значение ключа key, а также удалит его из словаря dict. Если ключ не найден, то вернет значение по умолчанию default.

```
>>> x = {'one': 0, 'two': 20, 'three': 3}
>>> x.pop('three')
3
>>> x
{'one': 0, 'two': 20}
>>> x.pop('three', 150)
150
>>> x.pop('three')
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# KeyError: 'ten'
```

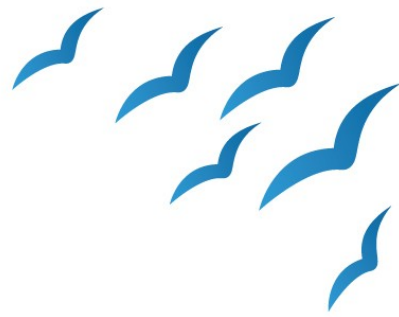


values()

Метод `dict.values()` возвращает новый список-представление всех значений `dict_values`, содержащихся в словаре `dict`.

Список-представление значений `dict_values`, является динамичным объектом. Это значит, что все изменения, такие как удаление, изменение или добавление значений в словаре сразу отражаются на этом представлении.

```
>>> x = {'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> values = x.values()
>>> values
# dict_values([1, 2, 3, 4])
```



popitem()

Метод `dict.popitem()` удалит и вернет двойной кортеж `(key, value)` из словаря `dict`. Пары возвращаются с конца словаря, в порядке **LIFO** (последним пришёл – первым ушёл)

```
>>> x = {'one': 0, 'two': 20, 'three': 3}
>>> x.popitem()
('four', 4)
>>> x.popitem()
('three', 3)
>>> x.popitem()
('two', 20)
>>> x.popitem()
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# KeyError: 'popitem(): dictionary is empty'
```

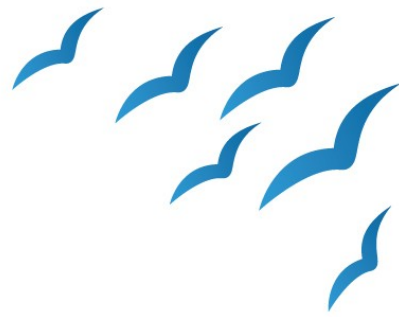
Множества set



Множество — неупорядоченный набор элементов. Каждый элемент в множестве уникален (т. е. повторяющихся элементов нет) и неизменяем.

```
>>> data_scientist =  
set(['Python', 'R', 'SQL', 'Pandas', 'Git'])
```

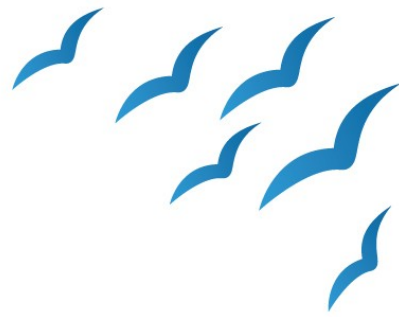
```
>> data_engineer =  
set(['Python', 'Java', 'Hadoop', 'SQL', 'Git' ])
```



Задание множества

Что будет при дублировании значения ?

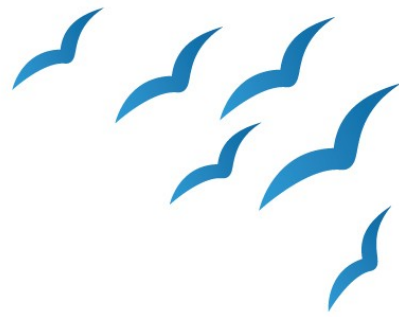
```
>>> data_scientist =  
set(['Python', 'R', 'R', 'SQL', 'Pandas', 'Git'])  
  
>>> type(data_scientist)  
<class 'set'>
```

Задание множества

Мы также можем создать множество с элементами разных типов. Например:

```
>>> mixed_set = {2.0, "Nicholas", (1, 2, 3)}  
>>> print(mixed_set)  
{'Nicholas', 2.0, (1, 2, 3)}
```



Задание множества

Мы также можем создать множество из списков.

```
>>> num_set = set([1, 2, 3, 4, 5, 6])  
>>> print(num_set)
```

Итерирование множества



```
months = {"Jan", "Feb", "March", "Apr",  
"May", "June", "July", "Aug", "Sep", "Oct",  
"Nov", "Dec"}
```

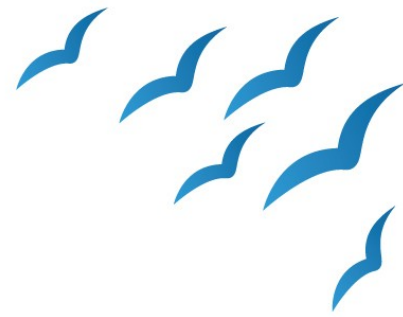
```
for m in months:
```

```
    print(m)
```

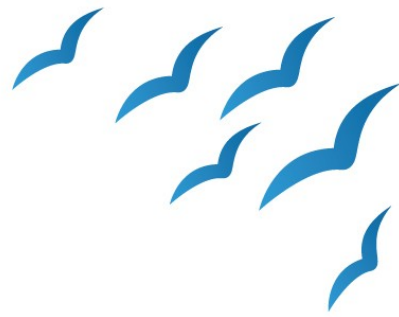
```
# проверка на членство в множестве
```

```
print("May" in months)
```

IMPORTANT METHODS IN PYTHON



SET	LIST	DICTIONARY
<ul style="list-style-type: none">• add()• clear()• pop()• union()• issuperset()• issubset()• intersection()• difference()• isdisjoint()• setdiscard()• copy()	<ul style="list-style-type: none">• append()• copy()• count()• insert()• reverse()• remove()• sort()• pop()• extend()• index()• clear()	<ul style="list-style-type: none">• copy()• clear()• fromkeys()• items()• get()• keys()• pop()• values()• update()• setdefault()• popitem()

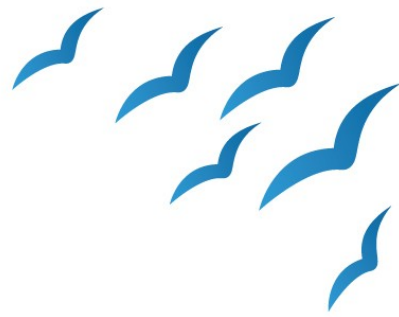


Добавление элементов

```
months = set(["Jan", "March", "Apr", "May",  
"June", "July", "Aug", "Sep", "Oct", "Nov",  
"Dec"])
```

```
months.add("Feb")
```

```
print(months)
```

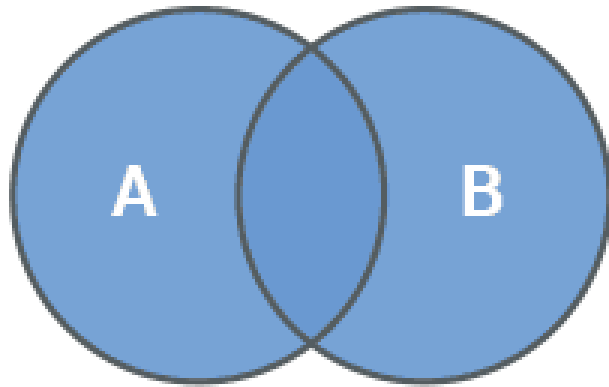
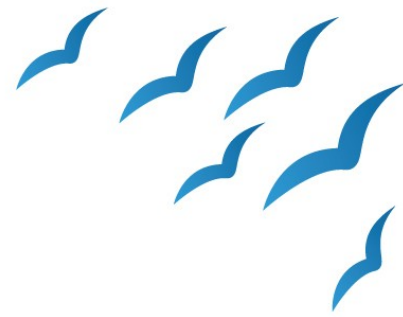


Удаление элемента из МНОЖЕСТВ

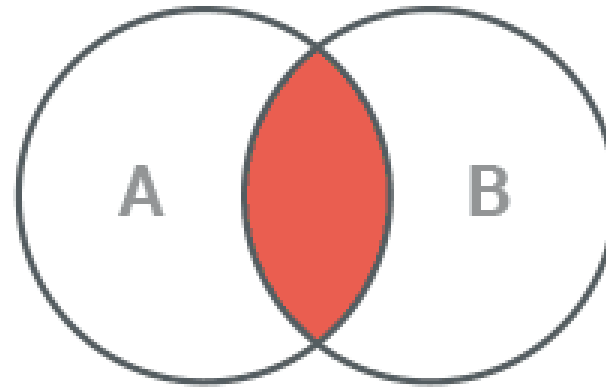
```
>>>num_set = {1, 2, 3, 4, 5, 6}  
>>>num_set.discard(3)  
>>>print(num_set)  
{1, 2, 4, 5, 6}
```

Метод `num_set.remove(7)`
аналогичный но вызовет ошибку при
отсутствии элемента.

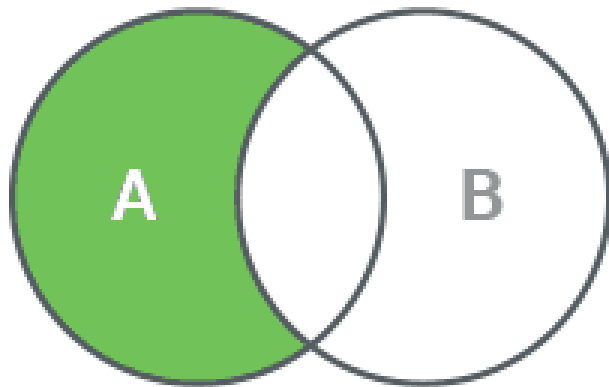
Из теории множеств



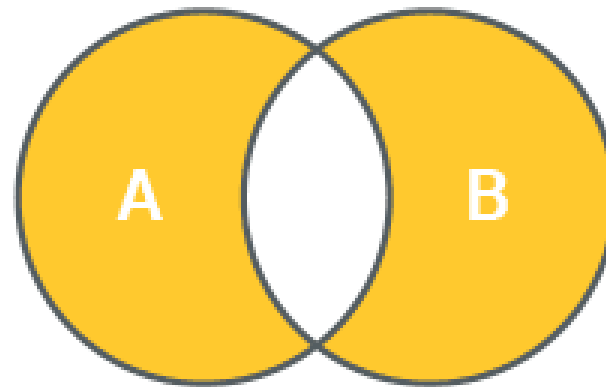
Union



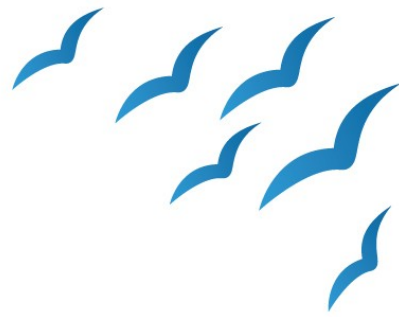
Intersection



Difference



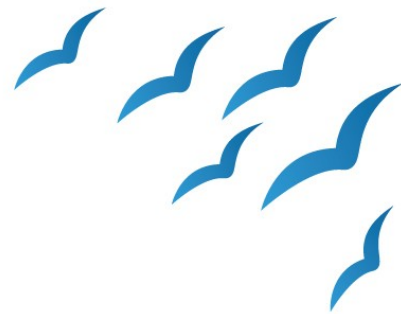
Symmetric Difference



Объединение множеств

```
>>> months_a = set(["Jan", "Feb", "March", "Apr",  
"May", "June"])  
>>> months_b = set(["July", "Aug", "Sep", "Oct",  
"Nov", "Dec"])  
  
>>> all_months = months_a.union(months_b)  
print(all_months)  
{'Oct', 'Jan', 'Nov', 'May', 'Aug', 'Feb', 'Sep',  
'March', 'Apr', 'Dec', 'June', 'July'}
```


union() или оператор |



Объединение может состоять из более чем двух множеств

```
x = {1, 2, 9}
```

```
y = {4, 5, 6}
```

```
z = {7, 8, 9}
```

```
output = x.union(y, z)
```

```
print(output)
```

```
Python
```

```
{1, 2, 9, 4, 5, 6, 7, 8}
```

```
print(x | y | z )
```

Пересечение множеств



```
x = {1, 2, 3}
y = {4, 3, 6}
z = x.intersection(y)
print(z) #
```

```
x = {1, 2, 3}
y = {4, 3, 6}
```

```
print(x & y)
```

3

Разница между множествами



```
set_a = {1, 2, 3, 4, 5}
```

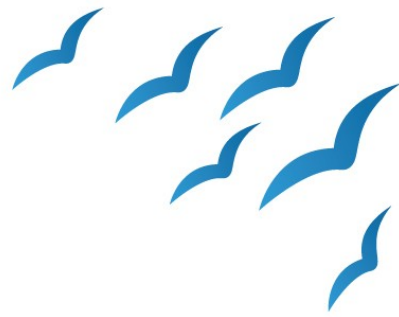
```
set_b = {4, 5, 6, 7, 8}
```

```
diff_set = set_a.difference(set_b)
```

```
print(diff_set)
```

```
{1, 2, 3}
```

```
print(set_a - set_b)
```

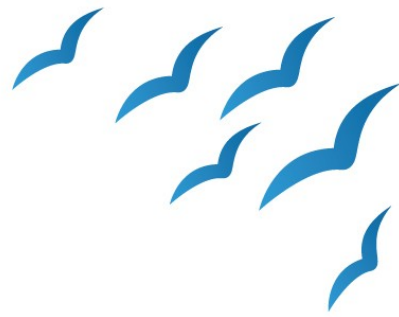


Симметричная разница

```
set_a = {1, 2, 3, 4, 5}
set_b = {4, 5, 6, 7, 8}
symm_diff = set_a.symmetric_difference(set_b)

print(symm_diff)
{1, 2, 3, 6, 7, 8}

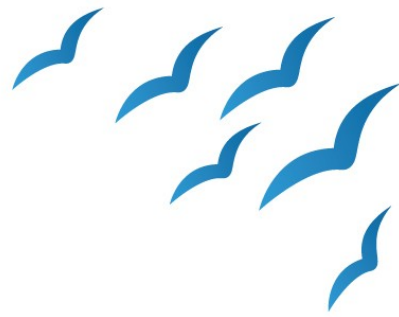
print(set_a ^ set_b)
```



Сравнение множеств

Чтобы проверить, является ли множество А дочерним от В, мы можем выполнить следующую операцию:

```
months_a = set(["Jan", "Feb", "March", "Apr", "May",  
"June"])  
months_b = set(["Jan", "Feb", "March", "Apr", "May",  
"June", "July", "Aug", "Sep", "Oct", "Nov", "Dec"])  
  
# Чтобы проверить, является ли множество В  
# подмножеством А  
subset_check = months_a.issubset(months_b)  
  
# Чтобы проверить, является ли множество А  
# родительским множеством  
superset_check = months_b.issuperset(months_a)  
  
print(subset_check)  
print(superset_check)
```



Метод `isdisjoint()`

Этот метод проверяет, является ли множество пересечением или нет. Если множества не содержат общих элементов, метод возвращает `True`, в противном случае — `False`.

```
names_a = {"Nicholas", "Michelle", "John",  
           "Mercy"}  
  
names_b = {"Jeff", "Bosco", "Teddy", "Milly"}  
  
x = names_a.isdisjoint(names_b)  
  
print(x)  
  
True
```

Frozenset в Python



Frozenset (замороженное множество) – это неизменные МНОЖЕСТВА.

```
X = frozenset([1, 2, 3, 4, 5, 6])
```

```
Y = frozenset([4, 5, 6, 7, 8, 9])
```

```
print(X)
```

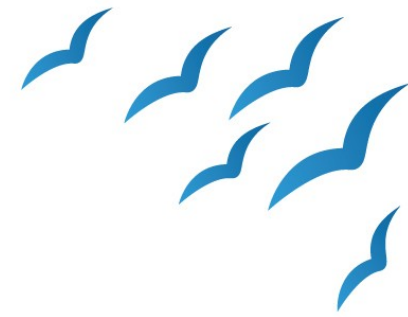
```
print(Y)
```

Какие методы для него не определены?

Картежи



- Они являются упорядоченными коллекциями произвольных объектов
- Поддержка доступа по индексу
- Неизменяемые последовательности
- Имеют фиксированную длину
- Представляют из себя массив ссылок на объекты



Упаковка кортежа

```
# пустой кортеж
```

```
empty_tuple = ()
```

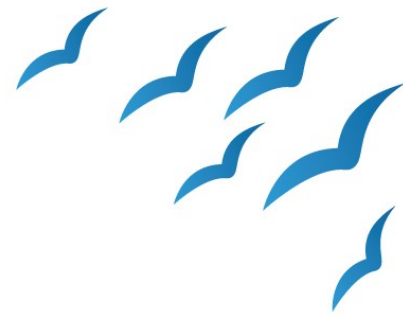
```
# кортеж из 4-х элементов разных типов
```

```
four_el_tuple = (36.6, 'Normal', None, False)
```

```
type(four_el_tuple)
```

```
<class 'tuple'>
```

Упаковка единственного элемента



```
tuple_one = ('a',)
```

```
tuple_two = 'b',
```

```
string_three = 'c'
```

```
print(type(is_tuple))
```

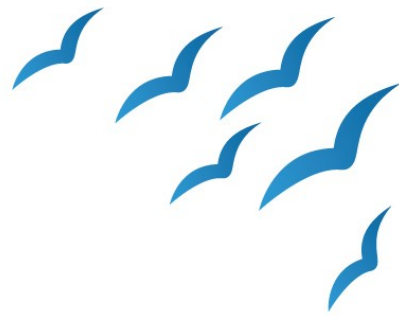
```
print(type(is_tuple_too))
```

```
print(type(string_three))
```

```
<class 'tuple'>
```

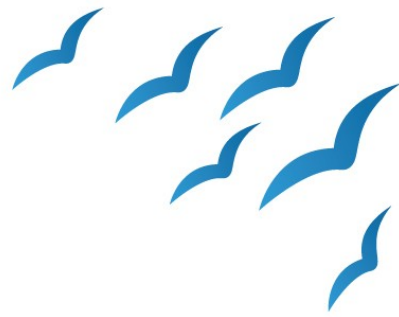
```
<class 'tuple'>
```

```
<class 'str'>
```



Природа множественного присваения

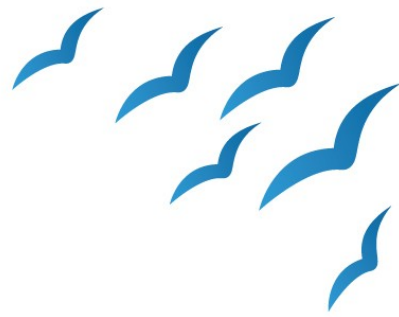
```
x, y = 100, 200
~
( x, y ) = (100, 200)
print(x)
100
print(y)
200
x, y = 'ML'      -   ?
```



Вложенные кортежи

пример tuple, что содержит вложенные элементы

```
nested_elem_tuple = (('one', 'two'),  
['three', 'four'], {'five': 'six'},  
(('seven', 'eight'), ('nine', 'ten')))  
print(nested_elem_tuple)
```



Множественное присвоение(распаковка)

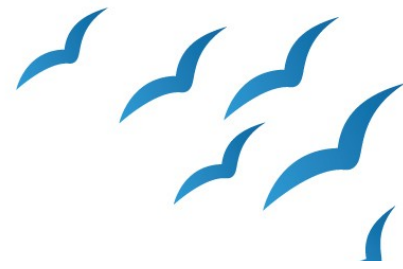
```
notes = ('Do', 'Re', 'Mi', 'Fa', 'Sol', 'La',  
         'Si')
```

```
do, re, mi, fa, sol, la, si = notes
```

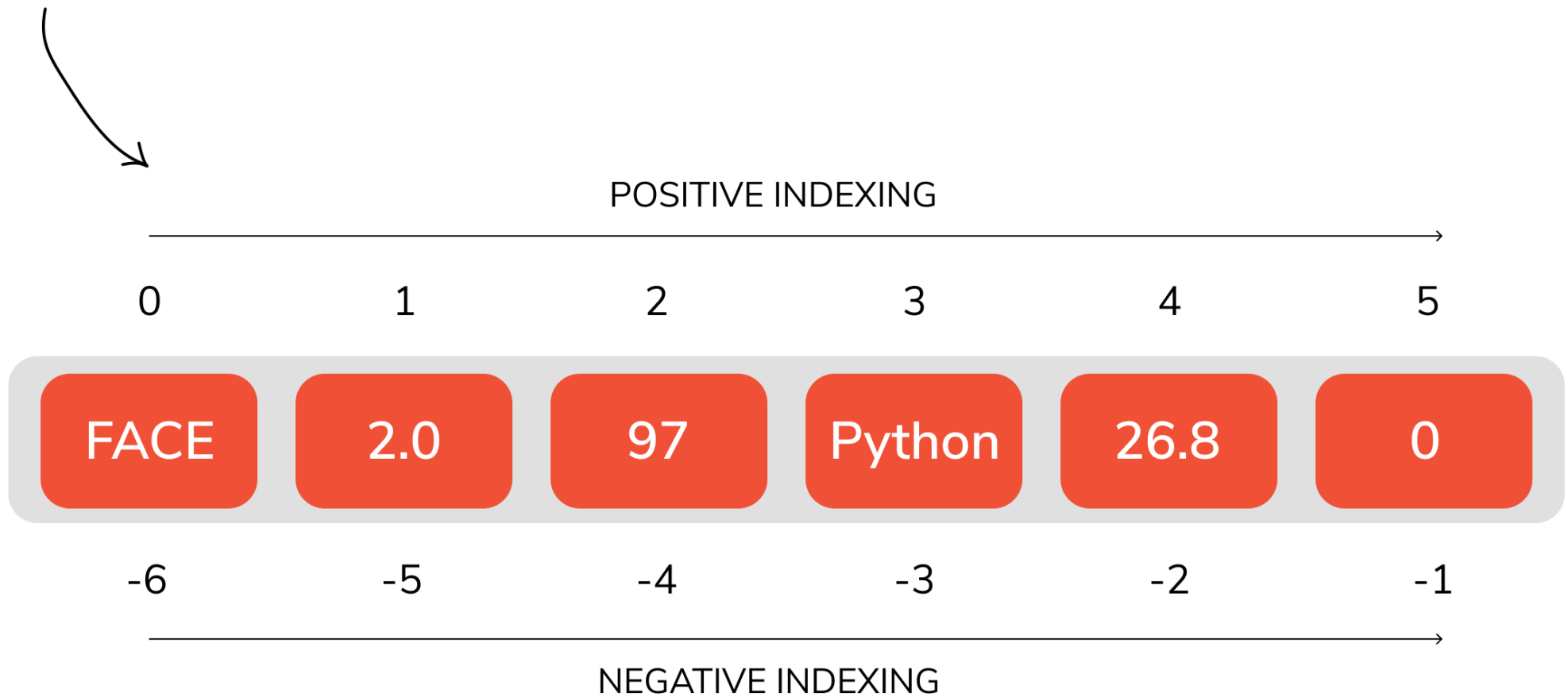
```
print(mi)
```

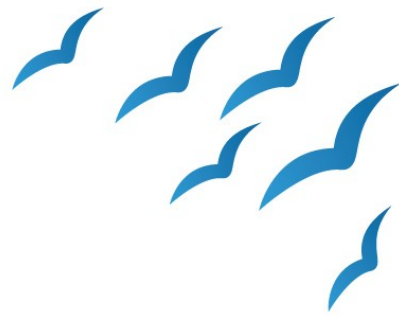
```
Mi
```

Индексация



Tuple = ('FACE', 2.0, 97, 'Python', 26.8, 0)



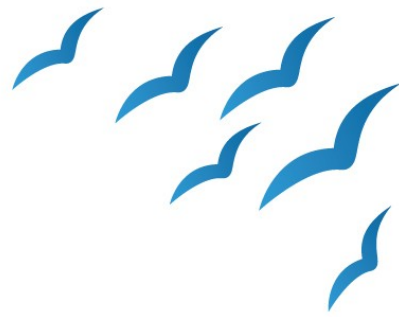


Доступ к элементам

```
>>> a = (1, 2, 3, 4, 5)
>>> print(a[0])
1
>>> print(a[1:3])
(2, 3)
```

Что произойдет ?

```
>>> a[1] = 3
```



Операции

Сложение

```
(1, 2) + (3, 4)
```

```
(1, 2, 3, 4)
```

Умножение

```
(1, 2) * 3
```

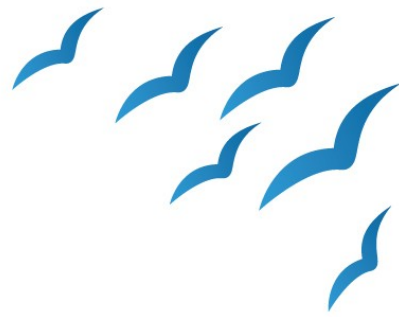
```
(1, 2, 1, 2, 1, 2)
```

Вхождение в кортеж

```
t_str = ('spam',)
```

```
'spam' in t_str
```

```
True
```

Сравнение

```
tuple_A = 2 * 2,  
tuple_B = 2 * 2 * 2,  
tuple_C = 'a',  
tuple_D = 'z',
```

```
# при сравнении кортежей, числа сравниваются по  
значению
```

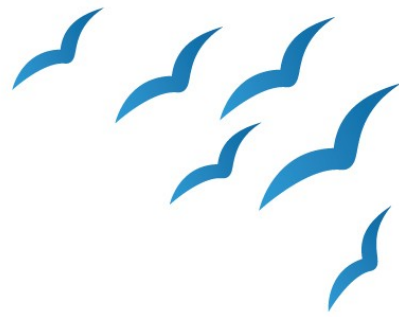
```
print(tuple_A < tuple_B)
```

```
> True
```

```
# строки в лексикографическом порядке
```

```
print(tuple_C < tuple_D)
```

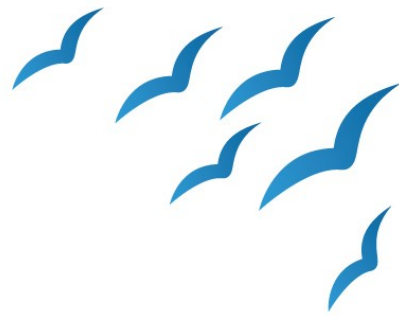
```
> True
```



Итерирование кортежа

```
language_token = ('if', 'for', 'while', 'list',  
'dict', 'tuple', 'set')
```

```
# Вывести все элементы кортежа  
for word in my_tuple:  
    print(word)
```



Сортировка

```
not_sorted_tuple = (10**5, 10**2, 10**1, 10**4,  
10**0, 10**3)
```

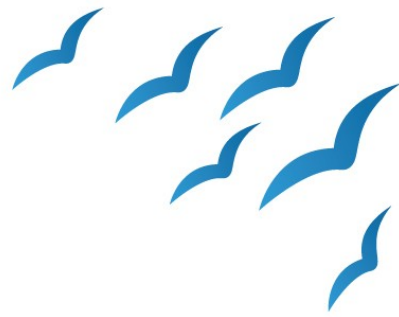
```
print(not_sorted_tuple)
```

```
> (100000, 100, 10, 10000, 1, 1000)
```

```
sorted_tuple = tuple(sorted(not_sorted_tuple))
```

```
print(sorted_tuple)
```

```
> (1, 10, 100, 1000, 10000, 100000)
```



Удаление

```
some_useless_stuff = ('sad', 'bad things', 'trans  
fats')
```

```
del some_useless_stuff
```

```
print(some_useless_stuff)
```

```
>
```

```
Traceback (most recent call last):
```

```
    print(some_useless_stuff)
```

```
NameError: name 'some_useless_stuff' is not  
defined
```

Срезы



Слайсы кортежей **`tuple[start:fin:step]`**

Где `start` — начальный элемент среза (включительно), `fin` — конечный (не включительно) и `step` — "шаг" среза.

```
float_tuple = (1.1, 0.5, 45.5, 33.33, 9.12, 3.14, 2.73)
```

```
print(float_tuple[0:3])
```

```
> (1.1, 0.5, 45.5)
```

```
# выведем элементы с шагом 2
```

```
print(float_tuple[-7::2])
```

```
> (1.1, 45.5, 9.12, 2.73)
```

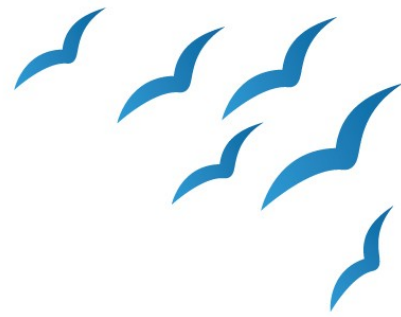
Python Tuple Methods

tuple.
 → count()
 → index()



Индекс заданного элемента

index(value, start, stop)



```
rom = ('I', 'II', 'III', 'IV', 'V', 'VI', 'VII',  
      'VIII', 'IX', 'X')
```

```
print(rom.index('X'))
```

9

```
str = ('aa', 'bb', 'aa', 'cc')
```

```
print(str.index('aa'))
```

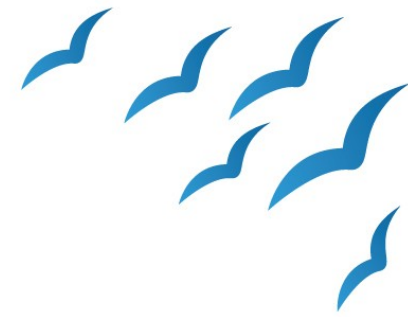
0

```
str = ('aa', 'bb', 'aa', 'cc' )
```

```
print(str.index('aa', 1, len(str)))
```

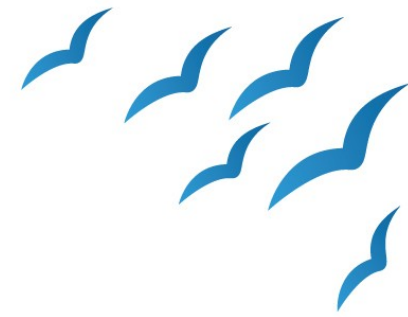
```
print(str.index('aa', 1,))
```

2



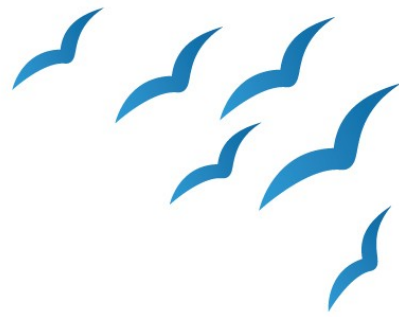
Число вхождений элемента **count()**

```
t_str = ('aa', 'bb', 'aa', 'cc')  
print(t_str.count('aa'))  
2
```

Длина кортежа **len()**

```
python_ = ('p', 'y', 't', 'h', 'o', 'n')  
print(len(python_))
```



Преобразование tuple → str

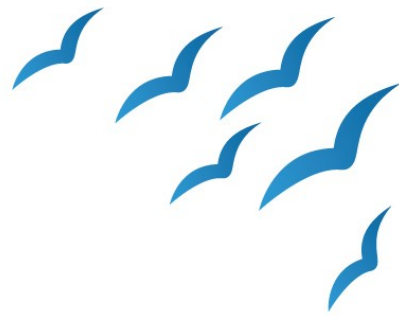
#Кортеж в строку

```
game_name = ('Breath', ' ', 'of', ' ', 'the',  
            ' ', 'Wild')
```

```
game_name = ''.join(game_name)
```

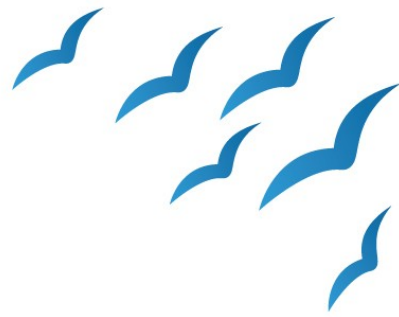
```
print(game_name)
```

```
Breath of the Wild
```



Преобразование tuple → list

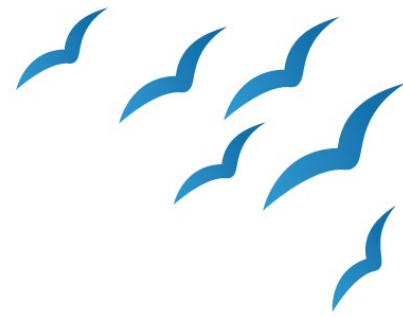
```
dig_tuple = (1111, 2222, 3333)
print(dig_tuple)
> (1111, 2222, 3333)
dig_list = list(dig_tuple)
print(dig_list)
[1111, 2222, 3333]
```



Преобразование tuple → dict

Преобразование через генератор
словарей

```
person = (('name', 'Piter' ), ('age', 100))  
p_dict = dict((x, y) for x, y in person)  
print(p_dict)  
{'name': 'Piter', 'age': 100}
```



Спасибо за внимание.

