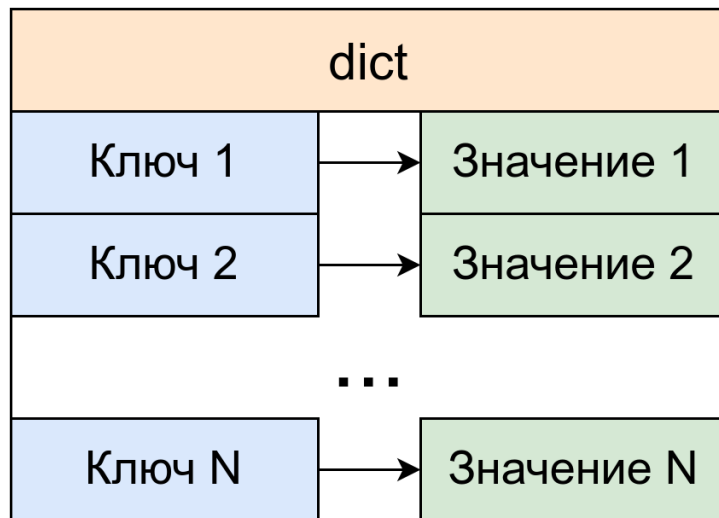


# Основы языка программирования Python <sup>1</sup>

---

**Словари  
(класс dict)**

**Словарь** (ассоциативный массив) — это структура данных, предназначенная для хранения элементов (значений), для доступа к которым используется *ключ*.



# Способы создания словарей

```
# Создание пустого словаря
```

```
foo = {}
```

```
# Создание заполненного словаря
```

```
bar = {  
    "key_1": 10,  
    "key_2": "Hello, world",  
    "key_3": [10, 20, 30]  
}
```

```
print("foo:", foo)
```

```
print("bar:", bar)
```

```
print("type(bar):", type(bar))
```

---

```
foo: {}
```

```
bar: {'key_1': 10, 'key_2': 'Hello, world', 'key_3': [10, 20, 30]}
```

```
type(bar): <class 'dict'>
```

```
foo = dict()
```

```
bar = dict(key_1=10,  
           key_2="Hello, world",  
           key_3=[10, 20, 30])
```

```
spam = dict([  
    ("key_1", 10),  
    ("key_2", "Hello, world"),  
    ("key_3", [10, 20, 30])  
])
```

```
print("foo:", foo)  
print("bar:", bar)  
print("spam:", spam)
```

---

```
foo: {}
```

```
bar: {'key_1': 10, 'key_2': 'Hello, world', 'key_3': [10, 20, 30]}
```

```
spam: {'key_1': 10, 'key_2': 'Hello, world', 'key_3': [10, 20, 30]}
```

```
keys = ["key_1", "key_2", "key_3"]  
values = [10, "Hello, world", [10, 20, 30]]  
  
foo = dict(zip(keys, values))  
  
print("foo:", foo)
```

---

```
foo: {'key_1': 10, 'key_2': 'Hello, world', 'key_3': [10, 20, 30]}
```

```
foo = [10, 15, 42, 50]
```

```
bar = {"key_" + str(n): n * 2 for n in foo}
```

```
print("bar:", bar)
```

---

```
bar: {'key_10': 20, 'key_15': 30, 'key_42': 84, 'key_50': 100}
```

```
foo = {}  
foo["key_1"] = 10  
foo["key_2"] = "Hello, world"  
foo["key_3"] = [10, 20, 30]  
foo["key_1"] = 200  
  
print(foo)
```

---

```
{'key_1': 200, 'key_2': 'Hello, world', 'key_3': [10, 20, 30]}
```



*# Ключи могут быть разного типа*

```
foo = {  
    "key_str": "Hello, world",  
    100: [10, 20, 30],  
    None: 0,  
}
```

```
print("foo:", foo)
```

---

```
foo: {'key_str': 'Hello, world', 100: [10, 20, 30], None: 0}
```

```
# Не все типы могут выступать в качестве ключей  
# Ошибка!  
foo = {[1, 2, 3]: "Hello, world"}
```

---

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: unhashable type: 'list'
```

# Использование словарей

```
print(dir(dict))
```

---

```
['__class__', '__class_getitem__', '__contains__', '__delattr__',  
 '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',  
 '__getattr__', '__getitem__', '__getstate__', '__gt__', '__hash__',  
 '__init__', '__init_subclass__', '__ior__', '__iter__', '__le__', '__len__',  
 '__lt__', '__ne__', '__new__', '__or__', '__reduce__', '__reduce_ex__',  
 '__repr__', '__reversed__', '__ror__', '__setattr__', '__setitem__',  
 '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys',  
 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

*# Получение элементов словаря*

```
foo = {  
    "key_1": 10,  
    "key_2": "Hello, dict",  
    "key_3": [10, 20, 30]  
}
```

```
print(foo["key_2"])
```

---

Hello, dict

*# Попытка получить отсутствующий элемент*

```
foo = {  
    "key_1": 10,  
    "key_2": "Hello, dict",  
    "key_3": [10, 20, 30]  
}
```

*# ! Ошибка*

```
print(foo["invalid_key"])
```

---

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

KeyError: 'invalid\_key'

*# Проверка наличия ключа*

```
foo = {"key_1": 10, "key_2": 20}  
key = "unknown_key"
```

```
if key in foo:  
    bar = foo[key]  
else:  
    bar = None
```

```
print("bar:", bar)
```

---

bar: None

*# Проверка наличия ключа*

```
foo = {"key_1": 10, "key_2": 20}  
key = "unknown_key"
```

```
bar = foo[key] if key in foo else None  
print("bar:", bar)
```

---

bar: None



*# Проверка наличия ключа*

```
foo = {"key_1": 10, "key_2": 20}
```

```
key = "unknown_key"
```

```
bar = None if key not in foo else foo[key]
```

```
print("bar:", bar)
```

---

bar: None

**get**(*key*[, *default*])

Return the value for *key* if *key* is in the dictionary, else *default*. If *default* is not given, it defaults to `None`, so that this method never raises a [KeyError](#).

---

```
foo = {"key_1": 10, "key_2": 20}
```

```
bar = foo.get("unknown_key")  
spam = foo.get("key_1")
```

```
print("bar:", bar)  
print("spam:", spam)
```

---

```
bar: None  
spam: 10
```

**get**(*key*[, *default*])

Return the value for *key* if *key* is in the dictionary, else *default*. If *default* is not given, it defaults to `None`, so that this method never raises a [KeyError](#).

---

```
foo = {"key_1": 10, "key_2": 20}
```

```
bar = foo.get("unknown_key", 0)
```

```
spam = foo.get("key_1", 0)
```

```
print("bar:", bar)
```

```
print("spam:", spam)
```

---

```
bar: 0
```

```
spam: 10
```

*# Добавление нового элемента в словарь  
# без перезаписи имеющегося значения*

```
foo = {"key_1": 10, "key_2": 20}
```

*# Данный ключ отсутствует в словаре*  

```
new_key = "new_key"  
new_value = 30
```

```
if new_key not in foo:  
    foo[new_key] = new_value  
    bar = new_value  
else:  
    bar = foo[new_key]
```

```
print("foo:", foo)  
print("bar:", bar)
```

---

```
foo: {'key_1': 10, 'key_2': 20, 'new_key': 30}  
bar: 30
```

*# Добавление нового элемента в словарь  
# без перезаписи имеющегося значения*

```
foo = {"key_1": 10, "key_2": 20}
```

*# Данный ключ имеется в словаре*

```
new_key = "key_1"
```

```
new_value = 30
```

```
if new_key not in foo:  
    foo[new_key] = new_value  
    bar = new_value
```

```
else:  
    bar = foo[new_key]
```

```
print("foo:", foo)
```

```
print("bar:", bar)
```

---

```
foo: {'key_1': 10, 'key_2': 20}
```

```
bar: 10
```

```
# Добавление нового элемента в словарь  
# без перезаписи имеющегося значения  
# Используем метод setdefault()
```

```
foo = {"key_1": 10, "key_2": 20}
```

```
# Данный ключ отсутствует в словаре  
new_key = "new_key"  
new_value = 30
```

```
bar = foo.setdefault(new_key, new_value)
```

```
print("foo:", foo)  
print("bar:", bar)
```

---

```
foo: {'key_1': 10, 'key_2': 20, 'new_key': 30}  
bar: 30
```

```
# Добавление нового элемента в словарь  
# без перезаписи имеющегося значения  
# Используем метод.setdefault()
```

```
foo = {"key_1": 10, "key_2": 20}
```

```
# Данный ключ имеется в словаре  
new_key = "key_1"  
new_value = 30
```

```
bar = foo.setdefault(new_key, new_value)
```

```
print("foo:", foo)  
print("bar:", bar)
```

---

```
foo: {'key_1': 10, 'key_2': 20}  
bar: 10
```

*# Удаление элементов*

```
foo = {  
    "key_1": 10,  
    "key_2": "Hello, dict",  
    "key_3": [10, 20, 30]  
}
```

```
del foo["key_3"]
```

```
print(foo)
```

---

```
{'key_1': 10, 'key_2': 'Hello, dict'}
```



*# Удаление всех элементов словаря*

```
foo = {"key_1": 10, "key_2": 20}  
foo.clear()
```

```
print("foo:", foo)
```

---

```
foo: {}
```

*# Перебор всех ключей словаря*

```
foo = {"key_1": 10, "key_2": 20, "key_3": 30}  
for key in foo.keys():  
    print(key)
```

---

key\_1

key\_2

key\_3

```
# Перебор всех значений словаря  
foo = {"key_1": 10, "key_2": 20, "key_3": 30}  
for key in foo.values():  
    print(key)
```

---

10

20

30

*# Одновременный перебор всех ключей и значений словаря*

```
foo = {"key_1": 10, "key_2": 20, "key_3": 30}
for key, value in foo.items():
    print(key, "->", value)
```

---

```
key_1 -> 10
key_2 -> 20
key_3 -> 30
```

```
foo = {"key_1": 10, "key_2": 20, "key_3": 30}  
keys = foo.keys()  
values = foo.values()  
items = foo.items()
```

```
print("type(keys)", type(keys))  
print("type(values)", type(values))  
print("type(items)", type(items))
```

---

```
type(keys) <class 'dict_keys'>  
type(values) <class 'dict_values'>  
type(items) <class 'dict_items'>
```

```
foo = {"key_1": 10, "key_2": 20, "key_3": 30}
keys = foo.keys()
values = foo.values()
items = foo.items()

print("keys:", keys)
print("values", values)
print("items", items)

foo["new_key"] = 100

print()
print("keys:", keys)
print("values", values)
print("items", items)
```

---

```
keys: dict_keys(['key_1', 'key_2', 'key_3'])
values dict_values([10, 20, 30])
items dict_items([('key_1', 10), ('key_2', 20), ('key_3', 30)])
```

```
keys: dict_keys(['key_1', 'key_2', 'key_3', 'new_key'])
values dict_values([10, 20, 30, 100])
items dict_items([('key_1', 10), ('key_2', 20), ('key_3', 30), ('new_key', 100)])
```

*# Обновление словаря на основе другого словаря*  
*# Реализация в виде цикла*

```
foo = {"key_1": 10, "key_2": 20, "key_3": 30}  
bar = {"key_1": 100, "key_2": 200, "key_other": 1000}  
  
for key, value in bar.items():  
    foo[key] = value  
  
print(foo)
```

---

```
{'key_1': 100, 'key_2': 200, 'key_3': 30, 'key_other': 1000}
```

*# Обновление словаря на основе другого словаря*  
*# Использование метода update()*

```
foo = {"key_1": 10, "key_2": 20, "key_3": 30}  
bar = {"key_1": 100, "key_2": 200, "key_other": 1000}  
  
foo.update(bar)  
print(foo)
```

---

```
{'key_1': 100, 'key_2': 200, 'key_3': 30, 'key_other': 1000}
```



`classmethod` **fromkeys**(*iterable*[, *value*])

Create a new dictionary with keys from *iterable* and values set to *value*.

[`fromkeys\(\)`](#) is a class method that returns a new dictionary. *value* defaults to `None`. All of the values refer to just a single instance, so it generally doesn't make sense for *value* to be a mutable object such as an empty list. To get distinct values, use a [dict comprehension](#) instead.

---

```
# Создание словаря на основе списка ключей  
# со значениями по умолчанию
```

```
keys = ["key_1", "key_2", "key_3"]
```

```
foo = dict.fromkeys(keys)  
bar = dict.fromkeys(keys, 0)
```

```
print("foo:", foo)  
print("bar:", bar)
```

---

```
foo: {'key_1': None, 'key_2': None, 'key_3': None}  
bar: {'key_1': 0, 'key_2': 0, 'key_3': 0}
```

*# Опасное использование значения по умолчанию*

```
keys = ["key_1", "key_2", "key_3"]
```

```
foo = dict.fromkeys(keys, [])
```

```
print("1) foo:", foo)
```

```
foo["key_1"].append(10)
```

```
print("2) foo:", foo)
```

---

1) foo: {'key\_1': [], 'key\_2': [], 'key\_3': []}

2) foo: {'key\_1': [10], 'key\_2': [10], 'key\_3': [10]}

*# Опасное использование значения по умолчанию*  
*# Пояснение к предыдущему коду*

```
keys = ["key_1", "key_2", "key_3"]
```

```
default = []
```

```
foo = dict.fromkeys(keys, default)
```

```
print("1) foo:", foo)
```

```
foo["key_1"].append(10)
```

```
print("2) foo:", foo)
```

---

1) foo: {'key\_1': [], 'key\_2': [], 'key\_3': []}

2) foo: {'key\_1': [10], 'key\_2': [10], 'key\_3': [10]}

## `copy()`

Return a shallow copy of the dictionary.

---

```
foo = {"key_1": 10, "key_2": 20, "key_3": 30}
bar = foo.copy()

print("bar:", bar)
```

---

```
bar: {'key_1': 10, 'key_2': 20, 'key_3': 30}
```

# Задачи

# Задача 1

Найдите три ключа с самыми большими значениями в словаре.

## Задача 2

Напишите программу для слияния нескольких словарей в один.

## Задача 3

Отсортируйте словарь по значению в порядке возрастания и убывания.