# Python: словари | Я.Шпора

Словарь (dict) — изменяемая коллекция. Каждый элемент словаря состоит из пары `ключ : з начение` .

## Как объявить словарь

### С помощью фигурных скобок

```
vegetable_yields_dict = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
    'Баклажаны': 2.8,
    'Перец': 2.2,
    'Капуста': 3.5
}
```

### С помощью функции `dict()`

Эта функция возвращает словарь. При её вызове без аргументов будет создан пустой словарь.

```
empty = dict()
print('Печатаем пустой словарь:', empty)
# Вывод в терминал: Печатаем пустой словарь: {}

vegetables = dict(
    Помидоры=6.5,
    Огурцы=4.3,
)

print('Печатаем словарь с овощами:', vegetables)
# Вывод в терминал: Печатаем словарь с овощами: {'Помидоры': 6.5, 'Огурцы
```

Другой вариант синтаксиса `dict()` : в качестве аргумента в эту функцию можно передать список кортежей (каждый из которых должен состоять из двух элементов), и эти кортежи будут преобразованы в элементы словаря.

```
vegetable_yields = dict(
    [
        ('Помидоры', 6.5),
        ('Огурцы', 4.3),
        ('Баклажаны', 2.8)
    ]
)
```

```
vegetable_yields = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
    'Баклажаны': 2.8
}
```

Если в каких-то кортежах, переданных в функцию `dict()`, совпадут первые элементы, в итоговый словарь попадёт последнее значение для дублирующегося ключа.

### С помощью функции dict.fromkeys(<keys>)

Функция принимает в качестве обязательного аргумента последовательность элементов и возвращает словарь. Значения элементов будут ключами словаря. Вторым, необязательным аргументом в функцию `dict.fromkeys()` можно передать значение, которое будет присвоено в качестве значения всем ключам словаря. По умолчанию второй аргумент равен `None`.

```
vegetables = dict.fromkeys(['Помидоры', 'Огурцы'])
print('Без второго аргумента:', vegetables)
# Вывод в терминал: {'Помидоры': None, 'Огурцы': None}

vegetable_yields = dict.fromkeys(['Ананасы', 'Брюква'], 3.2)
print('Вторым аргументом передали 3.2:', vegetable_yields)
# Вывод в терминал: {'Ананасы': 3.2, 'Брюква': 3.2}
```

## Работа со словарями

### Обращение к элементам словаря

Синтаксис обращения по ключу: `имя_словаря[ключ]` . Если ключ — это строка, то она должна быть заключена в кавычки.

```
vegetable_yields = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
    'Баклажаны': 2.8,
    'Перец': 2.2,
    'Капуста': 3.5
}

# Получить значение для ключа 'Баклажаны'.
print(vegetable_yields['Баклажаны'])
# Вывод в терминал: 2.8
```

## Проверка «есть ли ключ в словаре»

Для этой задачи используется оператор `in` , он работает так же, как и в других коллекциях.

```
vegetable_yields = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
    'Баклажаны': 2.8,
    'Перец': 2.2,
    'Капуста': 3.5
}

if 'Помидоры' in vegetable_yields:
    print('Помидоры есть!')
else:
    print('Помидоров нет!')

# Вывод в терминал: Помидоры есть!
```

## Функция dict.get(<dict>, key, default)

Возвращает из словаря значение по ключу. Если запрошенный ключ не найден, вернётся `None` или значение, переданное третьим, необязательным

аргументом.

```
vegetable_yields = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
    'Баклажаны': 2.8,
    'Перец': 2.2,
    'Капуста': 3.5
}

print(vegetable_yields.get('Баклажаны'))
# Вывод в терминал: 2.8
print(vegetable_yields.get('Томаты'))
# Вывод в терминал: None
print(vegetable_yields.get('Томаты', 100))
# Вывод в терминал: 100
```

## Добавление в словарь нового элемента и изменение существующего

Чтобы добавить элемент в словарь, нужно новому ключу словаря присвоить значение:

```
имя_словаря[новый_ключ] = значение
```

Таким же способом можно изменить значение для существующего ключа словаря:

```
vegetable_yields = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
    'Баклажаны': 2.8,
    'Перец': 2.2,
    'Капуста': 3.5
}

# Обновить значение.
vegetable_yields['Помидоры'] = 10
```

```
# Добавить новый ключ и значение.
vegetable_yields['Кабачки'] = 100
```

## Объединение словарей

Объединить словари можно функцией `dict.update(<dict_1>, <dict_2>)`. Она работает по принципу «добавить все элементы второго словаря в первый».

```
vegetable_yields = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
    'Баклажаны': 2.8
}

new_yields = {
    'Перец': 2.2,
    'Тыква': 3.4
}

dict.update(vegetable_yields, new_yields)
print(vegetable_yields)

# Вывод в терминал:
# {'Помидоры': 6.5, 'Огурцы': 4.3, 'Баклажаны': 2.8, 'Перец': 2.2, 'Тык
ва': 3.4}
```

## Удаление элемента из словаря

Для удаления элемента словаря применяется оператор `del`:

```
vegetable_yields = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
    'Баклажаны': 2.8
}

del vegetable_yields['Помидоры']
```

## Извлечение элемента из словаря

Чтобы извлечь элемент из словаря, нужно:

- получить элемент,
- удалить его из коллекции.

Для извлечения элемента из словаря применяется функция `dict.pop()`. Она принимает три аргумента:

- словарь, из которого извлекается элемент;
- ключ того элемента, значение которого нужно извлечь;
- необязательный аргумент — значение, которое вернёт функция в случае, если запрошенный ключ не найден в словаре.

Если третий аргумент не указан, а ключ в словаре не найден, возникнет ошибка `KeyError`.

```
vegetable_yields = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
    'Баклажаны': 2.8,
    'Перец': 2.2,
    'Тыква': 3.4
}

# При попытке получить значение элемента словаря с ключом 'Кабачки'
# функция dict.pop() вернёт значение по умолчанию - 'Название не найден
o!'
result = dict.pop(vegetable_yields, 'Кабачки', 'Название не найдено!')
print(result)
# Вывод в терминал: Название не найдено!

result = dict.pop(vegetable_yields, 'Баклажаны')
print('В переменную result сохранено значение', result)
# Вывод в терминал: 2.8

print(vegetable_yields)
# Вывод в терминал: {'Помидоры': 6.5, 'Огурцы': 4.3, 'Перец': 2.2, 'Тык
ва': 3.4}
```

## Получение всех ключей и значений элементов словаря

Функция `dict.keys(<dict>)` возвращает последовательность, хранящую ключи словаря.

Функция `dict.values(<dict>)` возвращает последовательность, элементами которой будут значения словаря.

Функция `dict.items(<dict>)` возвращает коллекцию типа `dict_items`. Каждый элемент этой коллекции — кортеж из двух элементов. Первый элемент каждого кортежа — это ключ определённого элемента словаря, второй элемент кортежа — это значение того же элемента словаря.

```
vegetable_yields = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
```

```
    'Баклажаны': 2.8
}

print(vegetable_yields.keys())
print(vegetable_yields.values())
print(vegetable_yields.items())

# Вывод в терминал:
# dict_keys(['Помидоры', 'Огурцы', 'Баклажаны'])
# dict_values([6.5, 4.3, 2.8])
# dict_items([('Помидоры', 6.5), ('Огурцы', 4.3), ('Баклажаны', 2.8)])
```

## Очистка словаря

Чтобы очистить словарь, нужно применить функцию `dict.clear(<dict>)` :

```
vegetable_yields = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
    'Баклажаны': 2.8,
    'Перец': 2.2,
    'Тыква': 3.4
}

dict.clear(vegetable_yields)
print(vegetable_yields)
# Вывод в терминал: {}
```

## Копирование словаря

Функция `dict.copy(<dict>)` создаёт копию исходного словаря:

```
vegetable_yields = {
    'Помидоры': 6.5,
    'Огурцы': 4.3,
    'Баклажаны': 2.8,
    'Перец': 2.2,
    'Тыква': 3.4
}
```

```
print('ID, на который ссылается vegetable_yields:', id(vegetable_yield
s))
# Вывод в терминал: ID, на который ссылается vegetable_yields: 23273679
27808

yield_copy = vegetable_yields.copy()
print('ID, на который ссылается yield_copy:', id(yield_copy))
# Вывод в терминал: ID, на который ссылается yield_copy: 2327369444544
```

## Неограниченное количество именованных параметров в функции: параметр **kwargs

```
def print_first_key_and_value(**kwargs):
    print('Тип переменной kwargs:', type(kwargs))
    print('Содержимое переменной kwargs:', kwargs)

    # kwargs - это словарь. Получить список ключей этого словаря...
    kwargs_keys = list(dict.keys(kwargs))
    # ...и список его значений.
    kwargs_values = list(dict.values(kwargs))

    # Напечатать первый ключ и первое значение.
    print(kwargs_keys[0], '-', kwargs_values[0])

    # Напечатать значение переменной,
    # которая даже не была объявлена в этой функции, но передана в аргуме
    print(kwargs['species'])

# Вызвать функцию, передав ей три именованных аргумента.
# Сама функция о таких именах ничего не знает, но готова их принять.
print_first_key_and_value(name='Помидор', species='Solanum lycopersicum',

# В параметр **kwargs можно передать любое количество именованных аргумен
print_first_key_and_value(species='Solanum melongena', name='Баклажан')

# Вывод в терминал:
# Тип переменной kwargs: <class 'dict'>
# Содержимое переменной kwargs: {
```

```
#     'name': 'Помидор',
#     'species': 'Solanum',
#     'origin': 'Южная Америка'
#     }
# name - Помидор
# Solanum
# Тип переменной kwargs: <class 'dict'>
# Содержимое переменной kwargs: {'species': 'Solanum melongena', 'name':
# species - Solanum melongena
# Solanum melongena
```

Различать, когда полагаться на шпаргалку, а когда на собственную память — вот знание мудрого.

*Тест Эмпирик, древнеримский экспериментатор*