# Знакомство со словарями

## Аннотация

*В этом уроке рассказывается о словарях — встроенной в Python мощной структуре данных. В других языках аналогичная структура называется map, HashMap, Dictionary.*

*Базовые функции работы со словарями показаны на простых примерах хранения библиотеки знаний о фильмах и актёрах.*

*В этом уроке рассматриваются только словари с ключами-строками.*

## Знакомство со словарями

Списки — это удобный и самый популярный способ сохранить большое количество данных в одной переменной. Списки индексируют все хранящиеся в них элементы. Первый элемент, как мы помним, лежит по индексу 0, второй по индексу 1 и так далее. Такой способ хранения позволяет быстро обращаться к элементу списка, зная его индекс.

```
actors = ['Джонни Депп', 'Эмма Уотсон', 'Билли Пайпер']
print(actors[1])
```

### Энциклопедия об актёрах

Представим, что мы делаем свою онлайн-энциклопедию (наподобие Википедии) об актёрах мирового кино. Для каждого актёра нужно сохранить текст статьи о нём. Названием статьи будет строка, состоящая из фамилии и имени актёра. Как правильно хранить такие данные?

Можно создать список кортежей. Каждый кортеж будет состоять из двух строк — названия и текста статьи.

```
actors = [('Джонни Депп', 'Джон Кристофер Депп Второй родился 9 июня 1963 года в Овенсб
оро, Кентукки..'),
 ('Сильвестр Сталлоне', 'Сильвестр Гарденцио Сталлоне родился в Нью-Йорке. Его отец, па
рикмахер Фрэнк Сталлоне — иммигрант из Сицилии, ...'),
 ('Эмма Уотсон', 'Эмма Шарлотта Дуерр Уотсон родилась в семье английских адвокатов. В п
ять лет переехала вместе с семьей из Парижа в Англию...'),
 # ...
 ]
```

Со временем количество статей значительно вырастет. Чтобы найти нужную статью по названию, нам придётся написать цикл `for`, который пройдёт по всем элементам списка `actors` и найдёт в нём кортеж, первый элемент которого равен искомому названию. В приведённом выше примере, чтобы найти статью об Эмме Уотсон, нам придётся в цикле пройти мимо Джонни Деппа и Сильвестра Сталлоне. Угадать заранее, что статья об Эмме Уотсон лежит после них, не получится.

Корень этой проблемы в том, что списки индексируются целыми числами. Мы же хотим находить информацию не по числу, а по строке — названию статьи. Было бы здорово, если бы индексами могли быть не числа, а строки. В списках это невозможно, однако возможно в словарях!

Словарь (в Python он называется **`dict`**) — это тип данных, позволяющий, как и список, хранить много данных. В отличие от списка в словаре для каждого элемента можно самому определить «индекс», по которому он будет доступен. Этот индекс называется ключом.

Вот пример создания словаря для энциклопедии об актёрах мирового кино:

```
actors = {
 'Джонни Депп': 'Джон Кристофер Депп Второй родился 9 июня 1963 года в Овенсборо, Кенту
кки..',
 'Сильвестр Сталлоне': 'Сильвестр Гарденцио Сталлоне родился в Нью-Йорке. Его отец, пар
икмахер Фрэнк Сталлоне — иммигрант из Сицилии, ...',
 'Эмма Уотсон': 'Эмма Шарлотта Дуерр Уотсон родилась в семье английских адвокатов. В пя
ть лет переехала вместе с семьей из Парижа в Англию...',
 # ...
 }
```

Элементы словаря перечисляются в фигурных скобках и разделяются запятой. До двоеточия указывается ключ, а после двоеточия — значение, доступное в словаре по этому ключу.

Пустой словарь можно создать двумя способами:

```
d = dict()
# или так
d = {}
```

После инициализации словаря мы можем быстро получать статью про конкретного актёра:

```
print(actors['Эмма Уотсон'])
```

Обращение к элементу словаря выглядит как обращение к элементу списка, только вместо целочисленного индекса используется ключ. В качестве ключа можно указать выражение — Python вычислит его значение, прежде чем обратится к искомому элементу.

```
first_name = 'Сильвестр'
last_name = 'Сталлоне'
print(actors[first_name + ' ' + last_name])
```

Если индекса в словаре нет, возникнет ошибка:

```
print(actors['Несуществующий ключ'])
KeyError: 'Несуществующий ключ'
```

Важная особенность словаря — его динамичность. Мы можем добавлять новые элементы, изменять их или удалять. Изменяются элементы точно так же, как в списках, только вместо целочисленного индекса в квадратных скобках указывается ключ:

```
actors['Эмма Уотсон'] = 'Новый текст статьи об Эмме Уотсон'
```

Также в словари можно добавлять и удалять новые элементы. Добавление синтаксически выглядит так же, как и изменение:

```
actors['Брэд Питт'] = 'Уильям Брэдли Питт, более известный как Брэд Питт — американский
актёр и продюсер. Лауреат премии «Золотой глобус» за 1995 год, ...';

del actors['Джонни Депп']
# больше в словаре нет ни ключа 'Джонни Депп', ни соответствующего ему значения

print(actors['Джонни Депп'])
KeyError: 'Джонни Депп'
```

Удалять элемент можно и по-другому:

```
actors.pop('Джонни Депп')
```

Единственное отличие этого способа от вызова **`del`** — он возвращает удалённое значение. Можно написать

```
deleted_value = actors.pop('Джонни Депп')
```

и в переменную deleted_value положится значение, которое хранилось в словаре по ключу 'Джонни Депп'. В остальном этот способ идентичен оператору **`del`**. В частности, если ключа 'Джонни Депп' в словаре нет, возникнет ошибка `KeyError`. Чтобы ошибка не появлялась, этому методу можно передать второй аргумент. Он будет возвращён, если указанного ключа в словаре нет. Это позволяет реализовать безопасное удаление элемента из словаря:

```
deleted_value = actors.pop('Джонни Депп', None)
```

Если ключа 'Джонни Депп' в словаре нет, в `deleted_value` попадёт `None`.

Оператор **`in`** позволяет проверить, есть ли ключ в словаре:

```
if 'Джонни Депп' in actors:
    print('У нас есть статья про Джонни Деппа')
```

Проверить, что ключа нет, можно с помощью аналогичного оператора **`not in`**:

```
if 'Сергей Безруков' not in actors:
    print('У нас нет статьи о Сергее Безрукове')
```