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 актерах.

1.Знакомство со словарями

Списки — 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 т. д. Такой способ хранения позволяет быстро обращаться к элементу списка, зная его индекс.

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

Представим, что мы делаем свою онлайн-энциклопедию об актерах мирового кино (наподобие Википедии). Для каждого актера нужно сохранить текст статьи о нем. Ее название — строка, состоящая из фамилии и имени актера. 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 Овенсборо, Кентукки...'),
 ('Сильвестр Сталлоне', 'Сильвестр Гарденцио Сталлоне родился в Нью-Йорке. '
                'Его отец, парикмахер Фрэнк Сталлоне — иммигрант из Сицилии...'),
 ('Эмма Уотсон', 'Эмма Шарлотта Дуерр Уотсон родилась в семье английских
адвокатов. '
                'В пять лет переехала вместе с семьей из Парижа в Англию...'),
 # ...
 ]
```

Со временем количество статей значительно вырастет. Чтобы найти нужную статью по названию, нам придется написать цикл for, который пройдет по всем элементам списка `actors` и найдет в нем кортеж, первый элемент которого равен искомому названию. В приведенном выше примере, чтобы найти статью об Эмме Уотсон, нам придется 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

Словарь (в Python он называется `dict`) — 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**.

2.Создание словаря

Вот пример создания словаря для энциклопедии об актерах мирового кино:

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

Создание словаря

Элементы словаря перечисляются в фигурных скобках (как и элементы множества!) 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 = {}
```

Вспомните, что создать пустое множество можно, только используя функцию `set()`. Теперь понятно, почему это так — пустые фигурные скобки зарезервированы для создания словаря.

3.Обращение к элементу словаря

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

```
print(actors['Эмма Уотсон'])
```

Важно!

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 ключа в словаре нет, возникнет ошибка:

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

```
KeyError: 'Несуществующий ключ'
```

4. Добавление и удаление элементов

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 новые элементы и удалять существующие.

Добавление элемента

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, ...'
```

Удаление элемента

Для удаления можно использовать инструкцию `del` (как и в списках):

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

print(actors['Джонни Депп'])

KeyError: 'Джонни Депп'
```

Удаление элемента

Удалять элемент можно и по-другому:

```
actors.pop('Джонни Депп')
```

Единственное отличие этого способа от вызова `del` — он возвращает удаленное значение. Можно написать так:

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

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.

Важно!

Чтобы ошибка не появлялась, этому методу можно передать второй аргумент. Он будет возвращен, 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` попадет None.

5.Проверка наличия элемента в словаре

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')
```

6. Нестроковые ключи

Решим следующую задачу. Пусть дан длинный список целых чисел `numbers`. Мы знаем, что некоторые числа встречаются в этом списке несколько раз. Нужно узнать, сколько именно раз встречается каждое из чисел.

```
numbers = [1, 10, 1, 6, 4, 10, 4, 2, 2, 1, 10, 1]
counts = {}
for number in numbers:
    if number not in counts:
        counts[number] = 1
    else:
        counts[number] += 1
```

Просто так сделать `counts[number] += 1` нельзя: если ключа `number` в словаре нет, возникнет ошибка KeyError.

В результате работы этой программы все элементы из списка `numbers` окажутся ключами словаря `counts`. Значением `counts[x]` будет количество раз, которое число `x` встретилось в списке `numbers`. Как это работает?

Цикл `for` перебирает все элементы списка `numbers` и для каждого проверяет, присутствует ли он уже в качестве ключа в `counts`. Если нет — значит, число встретилось нам впервые и мы инициализируем значение `counts[numbers] = 1`. Иначе увеличим `counts[number]` на единицу, поскольку число `number` встретилось нам повторно.

Почему для этой задачи не стоит использовать список, хотя ключи — обычные целые числа? Потому что, используя словарь, мы можем решить эту задачу и для вещественных чисел, и для очень больших целых чисел, и вообще для любых объектов, которые можно сравнивать.

7.Метод fromkeys создания словаря со значением по умолчанию

Метод `fromkeys(seq[, value])` создает новый словарь с ключами из `seq` и значениями из `value`. По умолчанию `value` присваивается значение None.

Если значение по умолчанию не задано:

```
d = dict.fromkeys(['key_1', 'key_2'])
print(d)
```

```
{'key_1': None, 'key_2': None}
```

Если значение задано:

```
d = dict.fromkeys(['key_1', 'key_2'], 42)
print(d)
```

```
{'key_1': 42, 'key_2': 42}
```

8. Метод get

Взять значение в словаре можно не только с помощью квадратных скобок, но и с помощью метода `get`:

```
article = actors.get('Джонни Депп')
```

Преимущество метода в том, что, кроме ключа, он может принимать и второй аргумент — значение, которое вернется, если заданного ключа нет:

```
article = actors.get('Джонни Депп', 'Статья о Джонни Деппа не найдена')
```

Воспользуемся этим приемом для улучшения нашей программы в задаче о повторяющихся числах:

```
numbers = [1, 10, 1, 6, 4, 10, 4, 2, 2, 1, 10, 1]
counts = {}
for number in numbers:
    counts[number] = counts.get(number, 0) + 1
```

Попробуйте понять, почему это работает верно.

9.Методы получения ключей, значений и пар (ключ, значение)

Все ключи словаря можно перебрать циклом `for`:

```
for actor_name in actors:
    print(actor_name, actors[actor_name])
```

Другой способ сделать то же самое — вызвать метод `.keys()`:

```
for actor_name in actors.keys():
    print(actor_name, actors[actor_name])
```

Метод .keys()

С помощью метода `.keys()` можно получить список всех ключей словаря:

```
actors_names = list(actors.keys())
```

Метод .values()

Есть и парный метод `.values()`, возвращающий все значения словаря:

```
all_articles = list(actors.values())
```

Он позволяет, например, проверить, есть ли какое-нибудь значение `value` среди значений словаря:

```
value in d.values()
```

Метод .items()

Если вы хотите перебрать элементы словаря `d` так, чтобы в переменной `key` оказывался ключ, а в `value` — соответствующее ему значение, это можно сделать с помощью метода `.items()` и цикла `for`.

```
for key, val in d.items():
```

Например:

```
for actor_name, article in actors.items():
    print(actor_name, article)
```