Тема 7. Словари (dictionary)

Словарь в **Python** является изменяемым типом данных, который, как и список, может содержать в себе любое количество объектов разного типа. По сути, он представляет собой ассоциативный массив. Каждый элемент имеет не номер, а ключ, т. е. элементы хранятся в виде пары «ключ: значение».

Общий вид словаря:

{<ключ 1>:<значение 1>, <ключ 2>:<значение 2>, … , <ключ N>:<значение N>}

Ключом может быть как строка, так и число.

Пример:

```
{1: 'Катя', 'age': 20}
```

Значения словаря хранятся в неотсортированном порядке. Словари хранят ссылки на объекты, а не сами объекты.

7.1. Создание словаря

Чтобы работать со словарем, его нужно создать. Сделать это можно несколькими способами.

1. С помощью литерала.

Пустой словарь:

```
a = {}
print(a)
```

```
{}
```

Словарь с заданными ключами и значениями:

```
a = {'name':'Катя', 'age':20}
print(a)
```

```
{'age': 20, 'name': 'Катя'}
```

Динамическое создание словаря:

```
a = {}
a['name'] = 'Катя'
a['age'] = 20
print(a)
```

```
{'name': 'Катя', 'age': 20}
```

Перед добавлением элементов нужно обязательно создать пустой словарь. Иначе интерпретатор выдаст ошибку.

2. С помощью функции dict.

При создании словаря таким способом ключи обязательно должны быть строками. Причем ключ не заключаем в кавычки. Данный способ позволяет создать словарь из пар значений.

```
a = dict(name='Катя', age=20, city='Новосибирск')
print(a)
```

```
{'age': 20, 'city': 'Новосибирск', 'name': 'Катя'}
```

Также словарь можно быстро создать из двух последовательностей используя рассмотренную в предыдущей главе функцию **zip()**.

```
a = ['name', 'age', 'city']
b = ['Катя', 20, 'Новосибирск']
c = dict(zip(a, b))
print(c)
```

```
{'age': 20, 'city': 'Новосибирск', 'name': 'Катя'}
```

3. С помощью fromkeys.

Создает словарь по списку ключей с пустыми значениями.

```
a = {}.fromkeys(['name', 'age'])
print(a)
```

```
{'name': None, 'age': None}
```

Также можно после запятой указать значение по умолчанию для всех полей.

```
a = {}.fromkeys(['name', 'age'], 20)
print(a)
```

```
{'name': 20, 'age': 20}
```

Однако таким способом нельзя указать разные значения по умолчанию для каждого ключа.

Например, если написать следующим образом:

```
a = {}.fromkeys(['name', 'age'], ['Катя', 20])
print(a)
```

в результате получим:

```
{'age': ['Катя', 20], 'name': ['Катя', 20]}
```

т. е. значением по умолчанию для каждого ключа будет одинаковый список.

4. С помощью генератора.

```
a = {i: i**2 for i in range(5)}
print(a)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

5. Заполнение словаря с клавиатуры с помощью генератора.

```
a = {input('Введите ключ: '):input('Введите значение: ') for i in range(2)}
print(a)
```

После запуска обратите внимание на порядок запроса ввода. Сначала запрашивается значение, а потом ключ.

```
Введите значение: Катя
Введите ключ: name
Введите значение: 20
Введите ключ: age
{'age': '20', 'name': 'Катя'}
```

7.2. Получение данных из словаря

Для вывода всего словаря используется функция **print()**. Данный способ мы использовали в предыдущем пункте.

Если же нужно вывести какой-то конкретный элемент словаря, то нужно знать его ключ. Можно вывести элемент, указав его ключ в квадратных скобках.

Пример:

```
a = {i: i**2 for i in range(5)}
print(a)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

При попытке получить значение из словаря по несуществующему ключу получим ошибку.

Пример:

```
a = {'name': 'Катя', 'age': 20}
print(a['city'])
```

```
builtins.KeyError: 'city'
```

7.3. Обновление данных в словаре

Обновить данные в словаре можно по ключу, указав пару: *ключ:новое значение*. Если такой ключ уже есть в словаре, то изменится соответствующее ему значение, а если такого ключа нет, то добавится новая пара *ключ:значение*.

Пример:

```
a = {'name': 'Катя', 'age': 20}
a['name'] = 'Коля'
print(a)
```

Так как ключ **'name'** в словаре уже есть, его значение изменится, и при выводе результата мы получим:

```
{'age': 20, 'name': 'Коля'}
```

А если напишем следующим образом:

```
a = {'name': 'Катя', 'age': 20}
a['city'] = 'Новосибирск'
print(a)
```

то, поскольку ключа **'city'** в словаре нет, добавится новое значение:

```
{'age': 20, 'city': 'Новосибирск', 'name': 'Катя'}
```

7.4. Удаление элементов словаря

При выполнении процедуры удаления можно удалить элемент по ключу, очистить весь словарь, или удалить полностью словарь вместе с его содержимым. Для этой процедуры используется оператор **del**.

Удаление по ключу

```
a = {'name': 'Катя', 'age': 20, 'city': 'Новосибирск'}
del a['name']
print(a)
```

```
{'city': 'Новосибирск', 'age': 20}
```

Если во время удаления указать несуществующий ключ, то получим ошибку.

Очистка словаря

```
a = {'name': 'Катя', 'age': 20}
a.clear()
print(a)
```

```
{}
```

Удаление всего словаря

```
a = {'name': 'Катя', 'age': 20}
del a
print(a)
```

```
builtins.NameError: name 'a' is not defined
```

т. е. словарь не найден, так как удалился полностью и объект «а» больше не существует.

7.5. Основные функции и методы словаря

В описании всех функций будем использовать один и тот же словарь:

a={'name': 'Катя', 'age': 20, 'city': 'Новосибирск'}

Обращаем особое внимание на то, что является функцией, а что методом, и как они используются.

- **len()** – возвращает количество записей в словаре;

```
print(len(a))
```
```
3
```

- **clear()** – очищает словарь;

```
a.clear()
print(a)
```
```
{}
```

- **copy()** – создает копию словаря;

```
b = a.copy()
print(b)
```
```
{'city': 'Новосибирск', 'age': 20, 'name': 'Катя'}
```

- **get(***key, default***)** – возвращает значение по ключу. Если такого ключа нет, то возвращает значение, указанное в параметре *default*. По умолчанию оно равно *None*;

Пример 1:

```
print(a.get('name'))
```
```
Катя
```

Пример 2:

```
print(a.get('name1', 'Ключ не найден'))
```
```
Ключ не найден
```

- **items()** – возвращает все пары *ключ: значение*;

```
print(a.items())
```
```
dict_items([('age', 20), ('name', 'Катя'), ('city', 'Новосибирск')])
```

- **keys() –** возвращает ключи в словаре;

```
print(a.keys())
```
```
dict_keys(['name', 'city', 'age'])
```

- **values() –** возвращает все значения в словаре;

```
print(a.values())
```
```
dict_values(['Катя', 20, 'Новосибирск'])
```

- **pop(***key[, default]***) –** удаляет ключ *key* из словаря и возвращает значение. Если ключ отсутствует в словаре, возвращает значение, указанное в *default*, а если *default* не указан, то вызывает исключение;

Пример 1:

```
print(a.pop('age'))
print(a)
```
```
20
{'name': 'Катя', 'city': 'Новосибирск'}
```

Пример 2:

```
print(a.pop('age1', 'Элемент отсутствует'))
```
```
Элемент отсутствует
```

Пример 3:

```
print(a.pop('age1'))
```
```
KeyError: 'age1'
```

- **popitem() –** удаляет из словаря и возвращает первую пару *ключ:значение*. Не забываем, что словарь не упорядочен, и после добавления порядок элементов может измениться. Если словарь не содержит элементов, то возвращает исключение.

```
a = {'name': 'Катя', 'age': 20, 'city': 'Новосибирск'}
b = a.popitem()
print(b)
print(a)
```
```
('name', 'Катя')
{'age': 20, 'city': 'Новосибирск'}
```

7.6. Перебор элементов словаря при помощи цикла

Перебрать все элементы списка можно благодаря циклу **for**, но словари Python не являются последовательностями. В качестве примера выведем элементы словаря двумя методами.

Первый способ использует метод keys(), возвращающий список всех ключей словаря.

```
a = dict(name='Катя', age=20, city='Новосибирск')
for i in a.keys():
    print(i, '-', a[i])
```

```
name - Катя
age - 20
city - Новосибирск
```

Второй способ: указываем словарь в качестве параметра. На каждой итерации цикла будет возвращаться ключ, с помощью которого внутри цикла можно получить значение, соответствующее этому ключу.

```
a = dict(name='Катя', age=20, city='Новосибирск')
for i in a:
    print(i, '-', a[i])
```

```
age - 20
city - Новосибирск
name - Катя
```