Словари

Введение в словари

Аннотация. В этом уроке мы начнем изучение словарей в Python, тип данных – `dict`. Этот тип данных похож на списки и применяется при решении многих задач.

Словари

В прошлых уроках мы изучили четыре типа коллекций в Python:

- списки – изменяемые коллекции элементов, индексируемые;
- строки – неизменяемые коллекции символов, индексируемые;
- кортежи – неизменяемые коллекции элементов, индексируемые;
- множества – изменяемые коллекции уникальных элементов, неиндексируемые.

Следующий тип – **словари** – изменяемые коллекции элементов **с произвольными индексами – ключами**. Если в списках элементы индексируются целыми числами, начиная с `0`, то в словарях — любыми ключами, в том числе в виде строк.

Как нам уже известно, списки — удобный и самый популярный способ хранения большого количества данных в одной переменной. Списки индексируют все хранящиеся в них элементы, что позволяет быстро обращаться к элементу, зная его индекс.

Приведенный ниже код:

```
languages = ['Python', 'C#', 'Java', 'C++']

print(languages[0])
print(languages[2])
```

выводит:

```
Python
Java
```

Допустим, мы хотим хранить имя создателя каждого языка программирования. Это можно сделать несколькими способами.

Способ 1. Хранить еще один список, где по соответствующему индексу будет находиться имя создателя языка программирования.

Приведенный ниже код:

```
languages = ['Python', 'C#', 'Java', 'C++']
creators = ['Гвидо ван Россум', 'Андерс Хейлсберг', 'Джеймс Гослинг', 'Бьёрн Страуструп']

print('Создателем языка', languages[0], 'является', creators[0])
```

выводит:

```
Создателем языка Python является Гвидо ван Россум
```

Подход рабочий, но хранить данные в двух коллекциях не очень удобно.

Способ 2. Хранить список кортежей с парами значений "язык - имя создателя" в каждом.

Приведенный ниже код:

```
languages = [('Python', 'Гвидо ван Россум'),
             ('C#', 'Андерс Хейлсберг'),
             ('Java', 'Джеймс Гослинг'),
             ('C++', 'Бьёрн Страуструп')]

print('Создателем языка', languages[2][0], 'является', languages[2][1])
```

выводит:

```
Создателем языка Java является Джеймс Гослинг
```

Тоже рабочий подход, однако не очень эффективный. Придется написать цикл `for` для поиска по всем элементам списка `languages` кортежа, первый элемент которого равен искомому (названию языка). Чтобы найти автора языка C++ , нужно будет в цикле пройти мимо Python, C# и Java. Не получится угадать заранее, что язык C++ лежит после них.

Приведенный ниже код:

```
languages = [('Python', 'Гвидо ван Россум'),
             ('C#', 'Андерс Хейлсберг'),
             ('Java', 'Джеймс Гослинг'),
             ('C++', 'Бьёрн Страуструп')]

for item in languages:
    if item[0] == 'C++':
        print('Создателем языка', item[0], 'является', item[1])
```

выводит:

```
Создателем языка C++ является Бьёрн Страуструп
```

Списки индексируются целыми числами, но в этом случае удобно было бы находить информацию не по числу, а по строке — названию языка программирования. В списках строки не могут быть индексами, однако в словарях это возможно.

Словарь (тип данных `dict`), как и список, позволяет хранить много данных. В отличие от списка, в словаре для каждого элемента можно произвольно определить «индекс» — **ключ**, по которому он будет доступен.

Словарь — реализация структуры данных "ассоциативный массив" или "хеш таблица". В других языках аналогичная структура называется `map`, `HashMap`, `Dictionary`.

Создание словаря

Чтобы создать словарь, нужно перечислить его элементы – пары ключ-значение – через запятую в фигурных скобках, как и элементы множества. Первым указывается ключ, после двоеточия — значение, доступное в словаре по этому ключу.

Приведенный ниже код:

```
languages = {'Python': 'Гвидо ван Россум',
```

```
        'C#': 'Андерс Хейлсберг',
        'Java': 'Джеймс Гослинг',
        'C++': 'Бьёрн Страуструп'}
```

создает словарь, в котором ключом служит строка — название языка программирования, а значением — имя создателя языка.

Обращение к элементу словаря

Извлечь значение элемента словаря можно, обратившись к нему по его ключу. Чтобы получить значение по заданному ключу, как и в списках, используем квадратные скобки `[]`, индексируем по ключу.

Способ 3. Приведенный ниже код:

```
languages = {'Python': 'Гвидо ван Россум',
        'C#': 'Андерс Хейлсберг',
        'Java': 'Джеймс Гослинг',
        'C++': 'Бьёрн Страуструп'}

print('Создателем языка C# является', languages['C#'])
```

выводит:

```
Создателем языка C# является Андерс Хейлсберг
```

В отличие от списков, номеров позиций в словарях нет.

Приведенный ниже код:

```
languages = {'Python': 'Гвидо ван Россум',
        'C#': 'Андерс Хейлсберг',
        'Java': 'Джеймс Гослинг',
        'C++': 'Бьёрн Страуструп'}

print('Создателем языка C# является', languages[1])
```

приводит к возникновению ошибки `KeyError`.

Ошибка `KeyError` возникнет и при попытке извлечь значение по несуществующему ключу. В качестве ключа можно указать выражение: Python вычислит его значение и обратится к искомому элементу.

Способ 4. Приведенный ниже код:

```
languages = {'Python': 'Гвидо ван Россум',
        'C#': 'Андерс Хейлсберг',
        'Java': 'Джеймс Гослинг',
        'C++': 'Бьёрн Страуструп'}

print('Создателем языка C# является', languages['C' + '#'])
```

выводит:

```
Создателем языка C# является Андерс Хейлсберг
```

Создание словаря с помощью функции dict()

Если ключи словаря — строки без каких-либо специальных символов, то для создания словаря можно использовать функцию `dict()`.

Приведенный ниже код:

```
info = dict(name = 'Timur', age = 28, job = 'Teacher')
```

создает словарь с тремя элементами, ключами которого служат строки `'name'`, `'age'`, `'job'`, а значениями – `'Timur'`, `28`, `'Teacher'`.

Создание словаря на основании списков и кортежей

Создавать словари можно на основе списков кортежей или кортежей списков. Первый элемент списка или кортежа станет ключом, второй — значением.

Приведенный ниже код:

```
info_list = [('name', 'Timur'), ('age', 28), ('job', 'Teacher')]  # список кортежей

info_dict = dict(info_list)  # создаем словарь на основе списка кортежей
```

создает словарь с тремя элементами, где ключи — строки `name`, `age`, `job`, а соответствующие им значения — `'Timur'`, `28`, `'Teacher'`.

Аналогично работает приведенный ниже код:

```
info_tuple = (['name', 'Timur'], ['age', 28], ['job', 'Teacher'])  # кортеж списков

info_dict = dict(info_tuple)  # создаем словарь на основе кортежа списков
```

Если необходимо создать словарь, каждому ключу которого соответствует одно и то же значение, можно воспользоваться методом `fromkeys()`.

Приведенный ниже код:

```
dict1 = dict.fromkeys(['name', 'age', 'job'], 'Missed information')
```

создает словарь с тремя элементами, где ключи — строки `'name'`, `'age'`, `'job'`, а соответствующие им значения: `'Missed information'`, `'Missed information'`, `'Missed information'`.

Если методу `fromkeys()` не передать второй параметр, то по умолчанию присваивается значение `None`.

Приведенный ниже код:

```
dict1 = dict.fromkeys(['name', 'age', 'job'])
```

создает словарь с тремя элементами, в которых ключи — строки `'name'`, `'age'`, `'job'`, а значения — `None`, `None`, `None`.

Пустой словарь

Пустой словарь можно создать двумя способами:

- с помощью пустых фигурных скобок;
- с помощью функции `dict()`.

Приведенный ниже код:

```
dict1 = {}
dict2 = dict()


print(dict1)
print(dict2)
print(type(dict1))
print(type(dict2))
```

создает два пустых словаря и выводит:

```
{}
{}
<class 'dict'>
<class 'dict'>
```

Вспомните, что создать пустое множество можно, только используя функцию `set()` , потому что пустые фигурные скобки зарезервированы для создания словаря.

Вывод словаря

Для вывода всего словаря можно использовать функцию `print()`:

```
languages = {'Python': 'Гвидо ван Россум',
             'C#': 'Андерс Хейлсберг',
             'Java': 'Джеймс Гослинг'}

info = dict(name = 'Timur', age = 28, job = 'Teacher')

print(languages)
print(info)
```

Функция `print()` выводит на экран элементы словаря в фигурных скобках, разделенные запятыми:

```
{'Python': 'Гвидо ван Россум', 'C#': 'Андерс Хейлсберг', 'Java': 'Джеймс Гослинг'}
{'name': 'Timur', 'age': 28, 'job': 'Teacher'}
```

Начиная с версии Python 3.6, словари являются упорядоченными, то есть сохраняют порядок следования ключей в порядке их внесения в словарь.

Примечания

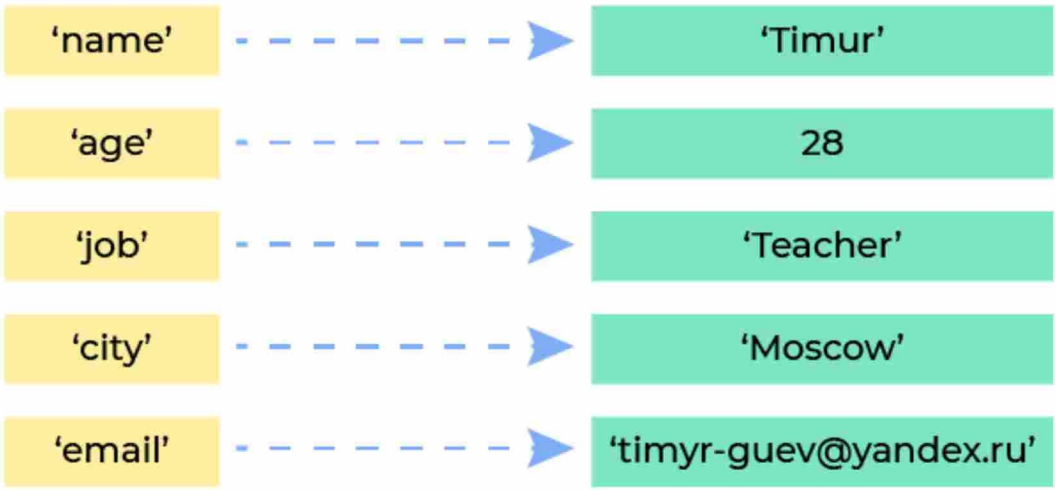
Примечание 1. Словари принципиально отличаются от списков по структуре хранения в памяти. Список — последовательная область памяти, то есть все его элементы (указатели на элементы) действительно хранятся в указанном порядке, расположены последовательно. Благодаря этому и можно быстро «прыгнуть» к элементу по его индексу. В словаре же используется специальная структура данных — **хеш-таблица**. Она позволяет вычислять

числовой хеш от ключа и использовать обычные списки, где в качестве индекса элемента берется этот хеш.

Примечание 2. В рамках одного словаря каждый ключ уникален.

Примечание 3. Словари удобно использовать для хранения различных сущностей. Например, если нужно работать с информацией о человеке, то можно хранить все необходимые сведения, включающие такие разные сущности как "возраст", "профессия", "название города", "адрес электронной почты" в одном словаре `info` и легко обращаться к его элементам по ключам:

```
info = {'name': 'Timur',
        'age': 28,
        'job': 'Teacher',
        'city': 'Moscow',
        'email': 'timyr-guev@yandex.ru'}

print(info['name'])
print(info['email'])
```



Примечание 4. Создать словарь на основании двух списков (кортежей) можно с помощью встроенной функции-упаковщика `zip()`, о которой расскажем позже.

Приведенный ниже код:

```
keys = ['name', 'age', 'job']
values = ['Timur', 28, 'Teacher']

info = dict(zip(keys, values))

print(info)
```

выводит (порядок элементов может отличаться):

```
{'name': 'Timur', 'age': 28, 'job': 'Teacher'}
```

В случае несовпадения длины списков функция самостоятельно отсечет лишние элементы.

Особенности словарей

Словари (тип данных `dict`) довольно просты, но о нескольких моментах следует помнить при их использовании.

Ключи должны быть уникальными

Словарь не может иметь два и более значения по одному и тому же ключу. Если при создании словаря (в литеральной форме) указать дважды один и тот же ключ, будет использовано последнее из указанных значений.

Приведенный ниже код:

```
info = {'name': 'Ruslan', 'age': 28, 'name': 'Timur'}

print(info['name'])
```

выводит:

```
Timur
```

Ключи должны быть неизменяемым типом данных

Ключом словаря могут быть данные любого неизменяемого типа:

- число;
- строка;
- булево значение;
- кортеж;
- замороженное множество (frozenset);
- ...

Приведенный ниже код создает словарь, ключами которого являются неизменяемые типы данных:

```
my_dict = {198: 'beegeek', 'name': 'Bob', True: 'a', (2, 2): 25}
```

Ключ словаря не может относиться к изменяемому типу данных:

- список;
- множество;
- словарь;
- ...

Приведенный ниже код приводит к возникновению ошибки:

```
my_dict = {[2, 2]: 25, {1, 2}: 'python', 'name': 'Bob'}
```

Значения могут относиться к любому типу данных, их тип данных произволен

Нет никаких ограничений для значений, хранящихся в словарях. Значения в словарях могут принадлежать к произвольному типу данных и повторяться для разных ключей многократно.

```
my_dict1 = {'a': [1, 2, 3], 'b': {1, 2, 3}}          # значения – изменяемый тип данных
```

```
my_dict2 = {'a': [1, 2], 'b': [1, 2], 'c': [1, 2]}    # значения повторяются
```

Основы работы со словарями

Основы работы со словарями

Работа со словарями похожа на работу со списками, поскольку и словари, и списки содержат в качестве отдельных элементов пары: в словарях `ключ: значение`, в списках `индекс: значение`.

Функция len()

Длиной словаря называется количество его элементов. Для определения длины словаря используют встроенную функцию `len()` (от слова length – длина).

Следующий программный код:

```
fruits = {'Apple': 70, 'Grape': 100, 'Banana': 80}
capitals = {'Россия': 'Москва', 'Франция': 'Париж'}

print(len(fruits))
print(len(capitals))
```

выведет:

```
3
2
```

Оператор принадлежности in

Оператор `in` позволяет проверить, содержит ли словарь заданный **ключ**.

Рассмотрим код:

```
capitals = {'Россия': 'Москва', 'Франция': 'Париж', 'Чехия': 'Прага'}

if 'Франция' in capitals:
    print('Столица Франции - это', capitals['Франция'])
```

Такой код проверяет, содержит ли словарь `capitals` элемент с ключом `Франция` и выводит соответствующий текст:

```
Столица Франции - это Париж
```

Можно использовать оператор `in` вместе с логическим оператором `not`.

Не забывайте, что при обращении по **несуществующему ключу**, возникнет ошибка во время выполнения программы.

Приведенный ниже код:

```
capitals = {'Россия': 'Москва', 'Франция': 'Париж', 'Чехия': 'Прага'}
```

```
print(capitals['Англия'])
```

приводит к возникновению ошибки:

```
KeyError: 'Англия'
```

Оператор принадлежности `in` на словарях работает **очень быстро**, намного быстрее, чем на списках, поэтому если нужен многократный поиск в коллекции данных, словарь – подходящий выбор.

Встроенные функции sum(), min(), max()

Встроенная функция `sum()` принимает в качестве аргумента **словарь с числовыми ключами** и вычисляет сумму его ключей.

Следующий программный код:

```
my_dict = {10: 'Россия', 20: 'США', 30: 'Франция'}

print('Сумма всех ключей словаря =', sum(my_dict))
```

выводит:

```
Сумма всех ключей словаря = 60
```

Для корректной работы функции `sum()` ключами словаря должны быть именно числа.

Встроенные функции `min()` и `max()` принимают в качестве аргумента словарь и находят минимальный и максимальный ключ соответственно, при этом ключ может принадлежать к любому типу данных, для которого возможны операции порядка <, <=, >, >= (числа, строки, и т.д.).

Приведенный ниже код:

```
capitals = {'Россия': 'Москва', 'Франция': 'Париж', 'Чехия': 'Прага'}
months = {1: 'Январь', 2: 'Февраль', 3: 'Март'}

print('Минимальный ключ =', min(capitals))
print('Максимальный ключ =', max(months))
```

выводит:

```
Минимальный ключ = Россия
Максимальный ключ = 3
```

Сравнение словарей

Словари можно сравнивать между собой. Равные словари имеют одинаковое количество элементов и содержат равные элементы (`ключ: значение`). Для сравнения словарей используются операторы `==` и `!=`.

Приведенный ниже код:

```
months1 = {1: 'Январь', 2: 'Февраль'}
```

```
months2 = {1: 'Январь', 2: 'Февраль', 3: 'Март'}
months3 = {3: 'Март', 1: 'Январь', 2: 'Февраль'}

print(months1 == months2)
print(months2 == months3)
print(months1 != months3)
```

выводит:

```
False
True
True
```

Примечания

Примечание 1. Обращение по индексу и срезы **недоступны** для словарей.

Примечание 2. Операция конкатенации + и умножения на число * **недоступны** для словарей.

Примечание 3. Словари нужно использовать в следующих случаях:

- Подсчет числа каких-то объектов. В этом случае нужно завести словарь, в котором ключи — названия объектов, а значения — их количество.
- Хранение каких-либо данных, связанных с объектом. Ключи — наименования объектов, значения — связанные с ними данные. Например, если нужно по названию месяца определить его порядковый номер, то это можно сделать при помощи словаря `num = {'January': 1, 'February': 2, 'March': 3, ...}`.
- Установка соответствия между объектами (например, “родитель-потомок”). Ключ — объект, значение — соответствующий ему объект.
- Если нужен обычный список, где максимальное значение индекса элемента очень велико, но при этом используются не все возможные индексы (так называемый “разреженный список”), то для экономии памяти можно использовать словарь.

Примечание 4. О том, как устроен словарь (тип `dict`) в Python можно почитать в статье.

Примечание 5. Исходный код словаря (тип `dict`) в Python можно найти тут.

Перебор элементов словаря

Перебор элементов словаря осуществляется так же, как перебор элементов списка – с помощью цикла `for`.

Для вывода ключей словаря **каждого на отдельной строке** можно использовать следующий код:

```
capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}

for key in capitals:
    print(key)
```

Такой код выведет (порядок элементов может отличаться):

```
Россия
Франция
Чехия
```

Для вывода значений словаря **каждого на отдельной строке** можно использовать следующий код:

```
capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}

for key in capitals:
    print(capitals[key])
```

Такой код выведет (порядок элементов может отличаться):

```
Москва
Париж
Прага
```

Для вывода элементов словаря **каждого на отдельной строке** можно использовать следующий код:

```
capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}

for key in capitals:
    print("Столица", key, "- это", capitals[key])
```

Такой код выведет (порядок элементов может отличаться):

```
Столица Россия - это Москва
Столица Франция - это Париж
Столица Чехия - это Прага
```

Методы keys(), values(), items()

Словарный метод `keys()` возвращает **список ключей** всех элементов словаря.

Следующий программный код:

```
capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}
```

```
for key in capitals.keys():  # итерируем по списку ["Россия", "Франция", "Чехия"]
    print(key)
```

выводит (порядок элементов может отличаться):

```
Россия
Франция
Чехия
```

Словарный метод `values()` возвращает **список значений** всех элементов словаря.

Следующий программный код:

```
capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}

for value in capitals.values():  # итерируем по списку ["Москва", "Париж", "Прага"]
    print(value)
```

выводит (порядок элементов может отличаться):

```
Москва
Париж
Прага
```

Словарный метод `items()` возвращает список всех элементов словаря, состоящий из кортежей пар `(ключ, значение)`.

Следующий программный код:

```
capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}

for item in capitals.items():
    print(item)
```

выводит (порядок элементов может отличаться):

```
("Россия", "Москва")
("Франция", "Париж")
("Чехия", "Прага")
```

Используя магию распаковки кортежей, можно писать такой код:

```
capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}

for key, value in capitals.items():
    print(key, "-", value)
```

Такой код выведет (порядок элементов может отличаться):

```
Россия - Москва
Франция - Париж
Чехия - Прага
```

Распаковка ключей словаря

Если требуется вывести только ключи словаря, то мы так же можем использовать операцию **распаковки словаря**.

Приведенный ниже код:

```
capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}

print(*capitals, sep="\n")
```

выводит (порядок элементов может отличаться):

```
Россия
Франция
Чехия
```

Начиная с версии Python 3.6, словари являются упорядоченными, то есть сохраняют порядок следования ключей в порядке их внесения в словарь.

Сортировка словаря

Так как словарь состоит из пар, то и отсортировать его можно как по ключам, так и по значениям.

Сортировка по ключам выполняется с использованием функции `sorted()`.

Приведенный ниже код:

```
capitals = {
    "Россия": "Москва",
    "Англия": "Лондон",
    "Чехия": "Прага",
    "Бразилия": "Бразилиа",
}

for key in sorted(capitals):
    print(key)
```

будет **гарантированно** выводить ключи словаря в алфавитном порядке, по возрастанию:

```
Англия
Бразилия
Россия
Чехия
```

Обратите внимание: функция `sorted()` возвращает отсортированный список ключей, а не словарь. Не путайте встроенную функцию `sorted()` и списочный метод `sort()`. Словари **не содержат** метода `sort()`.

Для сортировки словаря по значениям можно использовать функцию `sorted()` вместе с методом `items()`.

Приведенный ниже код:

```
capitals = {
    "Россия": "Москва",
    "Англия": "Лондон",
    "Чехия": "Прага",
    "Бразилия": "Бразилиа",
}

for key, value in sorted(capitals.items(), key=lambda x: x[1]):
    print(value)
```

будет **гарантированно** выводить значения словаря в алфавитном порядке, по возрастанию:

```
Бразилиа
Лондон
Москва
Прага
```

Стоит учитывать, что `sorted(capitals.items(), key=lambda x: x[1])` возвращает не словарь, а отсортированный по значениям список кортежей.

При сортировке словаря по значениям мы используем анонимную функцию `lambda x: x[1]`, о ней будет рассказано в следующих уроках.

Примечания

Примечание 1. Как мы уже знаем, с помощью оператора принадлежности `in` можно быстро проверить наличие ключа в словаре. Для проверки наличия значения в словаре можно использовать оператор `in` вместе с методом `values()`, который возвращает список всех значений словаря.

Проверка наличия ключа в словаре:

```
capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}

if "Россия" in capitals:
    print("В словаре есть ключ Россия")
```

Проверка наличия значения в словаре:

```
capitals = {"Россия": "Москва", "Франция": "Париж", "Чехия": "Прага"}

if "Париж" in capitals.values():
    print("В словаре есть значение Париж")
```

Примечание 2. Встроенная функция `sorted()` имеет опциональный параметр `reverse`. Если установить этому параметру значение `True`, то сортировка будет по убыванию.

Примечание 3. Код для работы со словарями нужно писать таким образом, чтобы он правильно работал при любом порядке обхода с помощью цикла `for`.

Примечание 4. Словарные методы `items()`, `keys()`, `values()` возвращают не совсем обычные списки. Типы этих списков – `dict_items`, `dict_keys`, `dict_values`, соответственно, в отличие от обычных списков `list`. Методы обычных списков недоступны

для списков типа `dict_items`, `dict_keys`, `dict_values`. Используйте явное преобразование с помощью функции `list()` для получения доступа к методам списков.

Методы словарей

Методы словарей

Словари, как и списки, имеют много полезных методов для упрощения работы с ними и решения повседневных задач. В прошлом уроке мы уже познакомились с тремя словарными методами:

- метод `items()` – возвращает словарные пары `ключ: значение`, как соответствующие им кортежи;
- метод `keys()` – возвращает список ключей словаря;
- метод `values()` – возвращает список значений словаря.

Добавление и изменение элементов в словаре

Чтобы изменить значение по определенному ключу в словаре, достаточно использовать индексацию вместе с оператором присваивания. При этом если ключ уже присутствует в словаре, его значение заменяется новым, если же ключ отсутствует – то в словарь будет добавлен новый элемент.

Приведенный ниже код:

```
info = {'name': 'Sam',
        'age': 28,
        'job': 'Teacher'}

info['name'] = 'Timur'                # изменяем значение по ключу name
info['email'] = 'timyr-guev@yandex.ru'  # добавляем в словарь элемент с ключом email

print(info)
```

выводит (порядок элементов может отличаться):

```
{'name': 'Timur', 'age': 28, 'job': 'Teacher', 'email': 'timyr-guev@yandex.ru'}
```

Обратите внимание на отличие в поведении словарей и списков:

- Если в списке `lst` нет элемента с индексом `7`, то попытка обращения к нему, например, с помощью строки кода `print(lst[7])` приведет к возникновению ошибки. И попытка присвоить ему значение `lst[7] = 100` тоже приведет к возникновению ошибки.
- Если в словаре `dct` нет элемента с ключом `name`, то попытка обращения к нему, например, с помощью строки кода `print(dct['name'])` приведет к возникновению ошибки. Однако попытка присвоить значение по отсутствующему ключу `dct['name'] = 'Timur'` ошибки не вызовет.

Решим следующую задачу: пусть задан список чисел `numbers`, где некоторые числа встречаются неоднократно. Нужно узнать, сколько именно раз встречается каждое из чисел.

```
numbers = [9, 8, 32, 1, 10, 1, 10, 23, 1, 4, 10, 4, 2, 2, 2, 2, 1, 10, 1, 2, 2, 32, 23, 23]
```

Первый код, который приходит в голову, имеет вид:

```
numbers = [9, 8, 32, 1, 10, 1, 10, 23, 1, 4, 10, 4, 2, 2, 2, 2, 1, 10, 1, 2, 2, 32, 23, 23]

result = {}
for num in numbers:
    result[num] += 1
```

Однако просто так сделать `result[num] += 1` нельзя, так как если ключа `num` в словаре еще нет, то возникнет ошибка `KeyError`.

Следующий программный код полностью решает поставленную задачу:

```
numbers = [9, 8, 32, 1, 10, 1, 10, 23, 1, 4, 10, 4, 2, 2, 2, 2, 1, 10, 1, 2, 2, 32, 23, 23]

result = {}
for num in numbers:
    if num not in result:
        result[num] = 1
    else:
        result[num] += 1
```

Цикл `for` перебирает все элементы списка `numbers` и для каждого проверяет, присутствует ли он уже в качестве ключа в словаре `result`. Если значение отсутствует, значит, число встретилось впервые и мы инициируем значение `result[num] = 1`. Если значение уже присутствует в словаре, увеличим `result[num]` на единицу.

Этот код можно улучшить с помощью метода `get()`.

Метод get()

Мы можем получить значение в словаре по ключу с помощью индексации, то есть квадратных скобок. Однако, как мы знаем, в случае отсутствия ключа будет происходить ошибка `KeyError`.

Приведенный ниже код:

```
info = {'name': 'Bob',
        'age': 25,
        'job': 'Dev'}

print(info['name'])
```

выводит:

```
Bob
```

Приведенный ниже код:

```
info = {'name': 'Bob',
        'age': 25,
        'job': 'Dev'}
```

```
print(info['salary'])
```

приводит к возникновению ошибки:

```
KeyError: 'salary'
```

Для того чтобы избежать возникновения ошибки в случае отсутствия ключа в словаре, можно использовать метод `get()`, способный кроме ключа принимать и второй аргумент — значение, которое вернется, если заданного ключа нет. Когда второй аргумент не указан, то метод в случае отсутствия ключа возвращает `None`.

Приведенный ниже код:

```
info = {'name': 'Bob',
        'age': 25,
        'job': 'Dev'}

item1 = info.get('salary')
item2 = info.get('salary', 'Информации о зарплате нет')

print(item1)
print(item2)
```

выводит:

```
None
Информации о зарплате нет
```

С помощью словарного метода `get()` можно упростить код в задаче о повторяющихся числах.

```
numbers = [9, 8, 32, 1, 10, 1, 10, 23, 1, 4, 10, 4, 2, 2, 2, 2, 1, 10, 1, 2, 2, 32, 23, 23]

result = {}
for num in numbers:
    result[num] = result.get(num, 0) + 1
```

Цикл `for` перебирает все элементы списка `numbers` и для каждого элемента с помощью метода `get()` мы получаем либо его значение из словаря `result`, либо значение по умолчанию, равное `0`. К данному значению прибавляется единица, и результат записывается обратно в словарь по нужному ключу.

Метод update()

Метод `update()` реализует своеобразную операцию конкатенации для словарей. Он объединяет ключи и значения одного словаря с ключами и значениями другого. При совпадении ключей в итоге сохранится значение словаря, указанного в качестве аргумента метода `update()`.

Приведенный ниже код:

```
info1 = {'name': 'Bob',
```

```
        'age': 25,
        'job': 'Dev'}

info2 = {'age': 30,
        'city': 'New York',
        'email': 'bob@web.com'}

info1.update(info2)

print(info1)
```

выводит (порядок элементов может отличаться):

```
{'name': 'Bob', 'age': 30, 'job': 'Dev', 'city': 'New York', 'email': 'bob@web.com'}
```

В Python 3.9 появились операторы `|` и `|=`, которые реализуют операцию конкатенации словарей.

Приведенный ниже код:

```
info1 = {'name': 'Bob',
        'age': 25,
        'job': 'Dev'}

info2 = {'age': 30,
        'city': 'New York',
        'email': 'bob@web.com'}

info1 |= info2

print(info1)
```

аналогичен предыдущему коду.

Метод setdefault()

Метод `setdefault()` позволяет получить значение из словаря по заданному ключу, автоматически добавляя элемент словаря, если он отсутствует.

Метод принимает два аргумента:

- `key`: ключ, значение по которому следует получить, если таковое имеется в словаре, либо создать.
- `default`: значение, которое будет использовано при добавлении нового элемента в словарь.

В зависимости от значений параметров `key` и `default` возможны следующие сценарии работы данного метода.

Сценарий 1. Если ключ `key` присутствует в словаре, то метод возвращает значение по заданному ключу (независимо от того, передан параметр `default` или нет).

Приведенный ниже код:

```
info = {'name': 'Bob',
```

```
      'age': 25}

name1 = info.setdefault('name')          # параметр default не задан
name2 = info.setdefault('name', 'Max')   # параметр default задан

print(name1)
print(name2)
```

выводит:

```
Bob
Bob
```

Сценарий 2. Если ключ `key` отсутствует в словаре, то метод вставляет переданное значение `default` по заданному ключу.

Приведенный ниже код:

```
info = {'name': 'Bob',
      'age': 25}

job = info.setdefault('job', 'Dev')
print(info)
print(job)
```

выводит:

```
{'name': 'Bob', 'age': 25, 'job': 'Dev'}
Dev
```

При этом если значение `default` не передано в метод, то вставится значение `None`.

Приведенный ниже код:

```
info = {'name': 'Bob',
      'age': 25}

job = info.setdefault('job')
print(info)
print(job)
```

выводит:

```
{'name': 'Bob', 'age': 25, 'job': None}
None
```

Удаление элементов из словаря

Существует несколько способов удаления элементов из словаря:

- оператор `del`;
- метод `pop()`;
- метод `popitem()`;
- метод `clear()`.

Оператор del

С помощью оператора `del` можно удалять элементы словаря по определенному ключу.

Следующий программный код:

```
info = {'name': 'Sam',
        'age': 28,
        'job': 'Teacher',
        'email': 'timyr-guev@yandex.ru'}

del info['email']    # удаляем элемент имеющий ключ email
del info['job']      # удаляем элемент имеющий ключ job

print(info)
```

выводит (порядок элементов может отличаться):

```
{'name': 'Sam', 'age': 28}
```

Если удаляемого ключа в словаре нет, возникнет ошибка `KeyError`.

Метод pop()

Оператор `del` удаляет из словаря элемент по заданному ключу вместе с его значением. Если требуется получить само значение удаляемого элемента, то нужен метод `pop()`.

Следующий программный код:

```
info = {'name': 'Sam',
        'age': 28,
        'job': 'Teacher',
        'email': 'timyr-guev@yandex.ru'}

email = info.pop('email')        # удаляем элемент по ключу email, возвращая его значение
job = info.pop('job')            # удаляем элемент по ключу job, возвращая его значение

print(email)
print(job)
print(info)
```

выводит:

```
timyr-guev@yandex.ru
Teacher
{'name': 'Sam', 'age': 28}
```

Единственное отличие этого способа удаления от оператора `del` — он возвращает удаленное значение. В остальном этот способ идентичен оператору `del`. В частности, если удаляемого ключа в словаре нет, возникнет ошибка `KeyError`.

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:

```
surname = info.pop('surname', None)
```

Если ключа `surname` в словаре нет, то в переменной `surname` будет храниться значение `None`.

Метод popitem()

Метод `popitem()` удаляет из словаря **последний добавленный элемент** и возвращает удаляемый элемент в виде кортежа `(ключ, значение)`.

Следующий программный код:

```
info = {'name': 'Bob',
    'age': 25,
    'job': 'Dev'}

info['surname'] = 'Sinclar'

item = info.popitem()

print(item)
print(info)
```

выводит:

```
('surname', 'Sinclar')
{'name': 'Bob', 'age': 25, 'job': 'Dev'}
```

В версиях Python ниже 3.6 метод `popitem()` удалял случайный элемент.

Метод clear()

Метод `clear()` удаляет все элементы из словаря.

Следующий программный код:

```
info = {'name': 'Bob',
       'age': 25,
       'job': 'Dev'}

info.clear()

print(info)
```

выведет:

```
{}
```

Метод copy()

Метод `copy()` создает **поверхностную** копию словаря.

Следующий программный код:

```
info = {'name': 'Bob',
       'age': 25,
```

```
        'job': 'Dev'}

info_copy = info.copy()
print(info_copy)
```

выведет:

```
{'name': 'Bob', 'age': 25, 'job': 'Dev'}
```

Не стоит путать копирование словаря (метод `copy()`) и присвоение новой переменной ссылки на старый словарь.

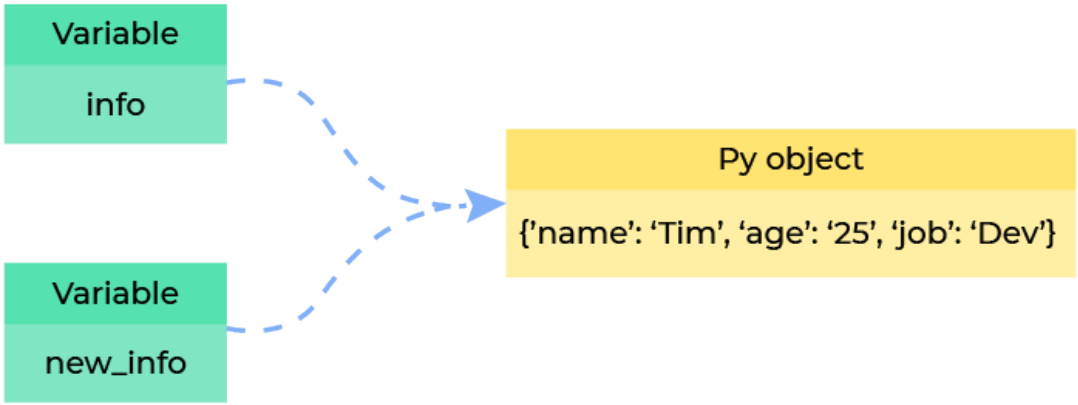
Следующий программный код:

```
info = {'name': 'Bob',
        'age': 25,
        'job': 'Dev'}

new_info = info
new_info['name'] = 'Tim'

print(info)
```

выводит:

```
{'name': 'Tim', 'age': 25, 'job': 'Dev'}
```

Оператор присваивания (=) не копирует словарь, а лишь присваивает ссылку на старый словарь новой переменной.



Таким образом, когда мы изменяем словарь `new_info`, меняется и словарь `info`. Если необходимо изменить один словарь, не изменяя второй, используют метод `copy()`.

Следующий программный код:

```
info = {'name': 'Bob',
        'age': 25,
```

```
        'job': 'Dev'}

new_info = info.copy()
new_info['name'] = 'Tim'

print(info)
print(new_info)
```

выводит:

```
{'name': 'Bob', 'age': 25, 'job': 'Dev'}
{'name': 'Tim', 'age': 25, 'job': 'Dev'}
```

Примечания

Примечание 1. Словарь можно использовать вместо нескольких вложенных условий, если вам нужно проверить число на равенство. Например, вместо

```
num = int(input())

if num == 1:
    description = 'One'
elif num == 2:
    description = 'Two'
elif num == 3:
    description = 'Three'
else:
    description = 'Unknown'

print(description)
```

можно написать

```
num = int(input())

description = {1: 'One', 2: 'Two', 3: 'Three'}

print(description.get(num, 'Unknown'))
```

На практике такой код встречается достаточно часто, особенно если в программе необходимо часто осуществлять проверку указанного типа.

Примечание 2. Методы `.get()` и `.setdefault()` в Python используются для работы со словарями, но они выполняют разные функции.

`.get(key, default)`:

- возвращает значение `default`, если ключ отсутствует в словаре (по умолчанию `None`, если `default` не указан);
- не изменяет сам словарь.

Пример:

```
my_dict = {'a': 1, 'b': 2}
```

value = my_dict.get('a', 0) # Получаем значение по ключу 'a', если ключ отсутствует, возвращаем 0

`.setdefault(key, default)`:

- если ключ отсутствует в словаре, вставляет новую пару ключ-значение с ключом `key` и значением `default` в словарь, затем возвращает `default`.

Пример:

my_dict = {'a': 1, 'b': 2}
value = my_dict.setdefault('c', 0) # Получаем значение по ключу 'c', если ключ отсутствует, вставляем 'c': 0

Важное отличие: `.get()` только получает значение, не меняя словарь, в то время как `.setdefault()` может изменить словарь, если ключ отсутствует.

Вложенные словари и генераторы словарей.

Вложенные словари

Словари могут содержать другие словари, которые сами, в свою очередь, содержат словари, и так далее на любую глубину. Такие словари называются **вложенными словарями** (мы уже сталкивались с вложенными списками и кортежами). Вложенные словари – один из способов представления структурированной информации.

Создание вложенных словарей

Вложенный словарь создается как обычный, только каждое значение в нем – другой словарь.

Приведенный ниже код создает словарь для хранения информации о сотрудниках некоторой компании.

```
info = {'emp1': {'name': 'Timur', 'job': 'Teacher'},
        'emp2': {'name': 'Ruslan', 'job': 'Developer'},
        'emp3': {'name': 'Rustam', 'job': 'Tester'}}
```

Тот же самый словарь info может быть создан по другому:

```
info = dict(emp1 = {'name': 'Timur', 'job': 'Teacher'},
            emp2 = {'name': 'Ruslan', 'job': 'Developer'},
            emp3 = {'name': 'Rustam', 'job': 'Tester'})
```

или

```
ids = ['emp1', 'emp2', 'emp3']

emp_info = [{'name': 'Timur', 'job': 'Teacher'},
            {'name': 'Ruslan', 'job': 'Developer'},
            {'name': 'Rustam', 'job': 'Tester'}]

info = dict(zip(ids, emp_info))
```

Число уровней вложенности словарей неограниченно!

Обращение к элементам вложенного словаря

Для того, чтобы получить значения определенных элементов во вложенном словаре, необходимо указать их ключи в нескольких квадратных скобках подобно тому, как мы получали значения во вложенных списках.

Приведенный ниже код:

```
info = {'emp1': {'name': 'Timur', 'job': 'Teacher'},
        'emp2': {'name': 'Ruslan', 'job': 'Developer'},
        'emp3': {'name': 'Rustam', 'job': 'Tester'}}

print(info['emp1']['name'])
print(info['emp2']['job'])
```

выводит:

```
Timur
Developer
```

При попытке обращения по несуществующему ключу возникнет ошибка KeyError. Для того, чтобы избежать возникновения такой ошибки, используйте словарный метод get(), который по умолчанию возвращает значение None, если ключ отсутствует в словаре.

Изменение вложенных словарей

Чтобы изменить значение определенного элемента во вложенном словаре, необходимо обратиться к его ключу и использовать оператор присвоения (=).

Приведенный ниже код:

```
info = {'emp1': {'name': 'Timur', 'job': 'Teacher'},
        'emp2': {'name': 'Ruslan', 'job': 'Developer'},
        'emp3': {'name': 'Rustam', 'job': 'Tester'}}

info['emp1']['job'] = 'Manager'

print(info['emp1'])
```

выводит:

```
{'name': 'Timur', 'job': 'Manager'}
```

Итерация по вложенным словарям

Итерации по вложенным словарям осуществляются как правило двумя циклами for (иногда достаточно одного цикла).

Приведенный ниже код:

```
info = {'emp1': {'name': 'Timur', 'job': 'Teacher'},
        'emp2': {'name': 'Ruslan', 'job': 'Developer'},
        'emp3': {'name': 'Rustam', 'job': 'Tester'}}

for emp in info:
    print('Employee ID:', emp)
    for key in info[emp]:
        print(key + ':', info[emp][key])
    print()
```

выводит:

```
Employee ID: emp1
name: Timur
job: Teacher

Employee ID: emp2
name: Ruslan
job: Developer
```

```
Employee ID: emp3
name: Rustam
job: Tester
```

Аналогичного результата можно было добиться с помощью метода items():

```
info = {'emp1': {'name': 'Timur', 'job': 'Teacher'},
        'emp2': {'name': 'Ruslan', 'job': 'Developer'},
        'emp3': {'name': 'Rustam', 'job': 'Tester'}}

for emp, inf in info.items():
    print('Employee ID:', emp)
    for key in inf:
        print(key + ':', inf[key])
    print()
```

Генераторы словарей

Пусть требуется создать словарь, ключами которого будут числа от 0 до 5, а значениями – квадраты ключей.

Для создания требуемого словаря можно использовать код:

```
squares = {}

squares[0] = 0
squares[1] = 1
squares[2] = 4
squares[3] = 9
squares[4] = 16
squares[5] = 25
```

или:

```
squares = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

или:

```
squares = {}

for i in range(6):
    squares[i] = i**2
```

Первые два способа слишком громоздкие и не подходят, когда чисел много. Третий способ полностью решает поставленную задачу, однако его можно переписать в стиле Python с использованием генератора словаря:

```
squares = {i: i**2 for i in range(6)}
```

Общий вид генератора словаря следующий:

{ключ: значение for переменная in последовательность}

где переменная — имя некоторой переменной, последовательность — последовательность значений, которые она принимает (любой итерируемый объект), ключ: значение — некоторое

выражение, как правило, зависящее от использованной в списочном выражении переменной, которой будут заполнены элементы словаря.

Примеры использования генератора словарей

Генераторы словаря могут выполнять итерации по любому итерируемому объекту: списки, кортежи, строки, словари.

1. Генератор словаря при итерировании по строке.

Приведенный ниже код:

```
dct = {c: c * 3 for c in 'ORANGE'}

print(dct)
```

выводит:

```
{'O': 'OOO', 'R': 'RRR', 'A': 'AAA', 'N': 'NNN', 'G': 'GGG', 'E': 'EEE'}
```

2. Для вычисления ключа и значения в генераторе словаря могут быть использованы выражения.

В следующем примере для вычисления ключа используется метод lower(), а для вычисления значения – метод upper().

Приведенный ниже код:

```
lst = ['ReD', 'GrEeN', 'BlUe']
dct = {c.lower(): c.upper() for c in lst}

print(dct)
```

выводит:

```
{'red': 'RED', 'green': 'GREEN', 'blue': 'BLUE'}
```

3. Извлечение из словаря элементов с определенными ключами.

Приведенный ниже код:

```
dict1 = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F'}
selected_keys = [0, 2, 5]

dict2 = {k: dict1[k] for k in selected_keys}


print(dict2)
```

выводит:

```
{0: 'A', 2: 'C', 5: 'F'}
```

Условия в генераторе словарей

В генераторах словарей можно использовать условный оператор.

Приведенный ниже код создает словарь, ключами которого являются четные числа от 0 до 9, а значениями – квадраты ключей:

```
squares = {}
for i in range(10):
    if i % 2 == 0:
        squares[i] = i**2

print(squares)
```

Такой код можно переписать с помощью генератора словаря в виде:

```
squares = {i: i**2 for i in range(10) if i % 2 == 0}

print(squares)
```

Не забываем про возможность указания шага в функции range(). Предыдущий код можно записать и без условного оператора: squares = {i: i**2 for i in range(0, 10, 2)}.

Генераторы вложенных словарей

Мы также можем использовать генераторы словарей для создания вложенных словарей:

Приведенный ниже код:

```
squares = {i: {j: j**2 for j in range(i + 1)} for i in range(5)}

for value in squares.values():
    print(value)
```

выводит:

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