**Словари в Python**

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])

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

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

languages = [('Python', 'Гвидо ван Россум'),

('C#', 'Андерс Хейлсберг'),

('Java', 'Джеймс Гослинг'),

('C++', 'Бьёрн Страуструп')]

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

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. Угадать заранее, что язык C++ лежит после них, не получится.

for item in languages:

if item[0] == 'C++':

print('Создателем языка', item[0], 'является', item[1])

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.

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.

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

'C#': 'Андерс Хейлсберг',

'Java': 'Джеймс Гослинг',

'C++': 'Бьёрн Страуструп'}

### 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#'])

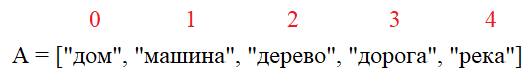
***Словарь****неупорядоченная коллекция произвольных данных с доступом по ключу.*

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

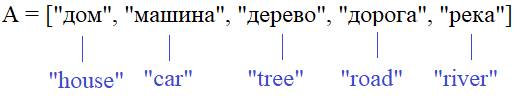
Например, в качестве ключа могут выступать обычные уникальные числа: 0, 1, 2, …

В этом случае ключи напоминают индексы элементов упорядоченного списка:

A = ["дом", "машина", "дерево", "дорога", "река"]



Но в списке изменить эти индексы мы не можем – они создаются списком автоматически. А, что если мы хотим ассоциировать эти слова со словами английского языка:



Как раз здесь нам на помощь и приходит словарь. То есть, мы вместо ключей 0, 1, 2,… создадим другие – в виде английских слов.

Для создания словаря используется следующий синтаксис: {key1: value1, key2: value2, …, keyN:valueN}

И для нашего случая, имеем:

d = {"house": "дом", "car": "машина",

"tree": "дерево", "road": "дорога",

"river":"река"}

Как видите, мы можем определять словарь в несколько строчек. Не обязательно все писать в одну строку.

Теперь, мы можем по ключу получать значение этого словаря по такому же синтаксису, что и для списков: Получим значение «дом». Если же указать не существующий ключ: то получим ошибку.

d["house"]

d[100]

Разумеется, ключи в словарях всегда уникальны. Если записать два одинаковых: то ключ «house» будет ассоциирован с последним указанным значением.

Словарь еще можно определить с помощью функции-конструктора dict:

d2 = dict(house = "дом", car = "машина",

tree = "дерево", road = "дорога", river = "река")

Здесь ключи записываются уже без кавычек и после них ставится знак равно вместо двоеточий. В результате, получаем точно такой же словарь. И такой способ задания работает только, если в качестве ключей выступают строки. Например, с числами такой подход работать не будет:

d2 = dict(1 = "дом", 2 = "машина")

Тогда зачем нужен такой способ: первый записывался короче и обладает большей гибкостью? Дело в том, что часто в программах приходится создавать словари на основе ранее сформированных данных, в частности, списков. Допустим, у нас имеется вот такой список:

lst = [[2, "неудовлетворительно"], [3, "удовлетворительно"], [4, "хорошо"], [5, "отлично"]]

Здесь у нас перечислены вложенные списки с двумя элементами: ключ, значение. Этот список можно преобразовать в словарь как раз с помощью функции dict:

d3 = dict(lst)

У объекта dict есть один интересный метод: dict.fromkeys(список[, значение по умолчанию]) который формирует словарь где ключами будут элементы списка. Например:

a = dict.fromkeys(["+7", "+6", "+5", "+4"])

Получим следующее:

{'+7': None, '+6': None, '+5': None, '+4': None}

все значения ключей равны None. Чтобы указать другое значение, используется второй необязательный аргумент:

a = dict.fromkeys(["+7", "+6", "+5", "+4"], "код страны")

получим словарь:

{'+7': 'код страны', '+6': 'код страны', '+5': 'код страны', '+4': 'код страны'}

Далее, если нужно создать пустой словарь, то это проще всего сделать так: или же так:

d = {}

d = dict()

Во всех наших примерах в качестве ключей словаря выступало либо число, либо строка. Какие типы данных еще можно указывать? На самом деле любые неизменяемые типы. Обратите внимание, именно **неизменяемые**! Например, можно написать так:

d[True] = "Истина"

d[False] = "Ложь"

Здесь ключи – булевые значения. В результате, получим словарь:

{True: 'Истина', False: 'Ложь'}

И этот пример показывает, что присваивая словарю значение с новым ключом, оно автоматически добавляется в словарь. В результате, наш изначально пустой словарь стал содержать две записи. Если же мы существующему ключу присваиваем другое значение:

d[True] = 1

то он просто поменяет свое значение в словаре. Вот так можно добавлять и менять значения словаря. То есть, словарь относится к изменяемым типам.

На значения словаря никаких ограничений не накладывается – там могут быть произвольные данные:

d = {True: 1, False: "Ложь", "list": [1,2,3], 5: 5}

Для удаления ключа можно воспользоваться оператором del:

**del** d[True]

Теперь записи с ключом True в словаре не будет. Однако, если при удалении будет указан несуществующий ключ: то возникнет ошибка. Поэтому перед удалением, можно проверить: существует ли данный ключ в словаре: Оператор in возвращает True, если ключ присутствует и False, если отсутствует. Или же, можно делать противоположную проверку на отсутствие ключа:

**del** d["abc"]

"abc" **in** d

"abc" **not** **in** d

Далее, как и для любой коллекции, мы можем определить длину (число записей) в словаре с помощью функции len:

len(d)

А также перебрать его значения с помощью цикла for:

d = {True: 1, False: "Ложь", "list": [1,2,3], 5: 5}

**for** x **in** d:

**print**(x)

И мы видим, что переменная x ссылается на значения ключей. Но, зная ключи, мы можем получить ассоциированное с ним значение:

**print**(x, d[x])

# Основные методы словарей

Итак, для удаления всех записей из словаря используется метод

d.**clear**()

Для создания копии словаря используется метод **copy**:

d = {True: 1, False: "Ложь", "list": [1,2,3], 5: 5}

d2 = d.copy()

d2["list"] = [5,6,7]

**print**(d)

**print**(d2)

Следующий метод **get** позволяет получать значение словаря по ключу: Его отличие от оператора в том, что при указании неверного ключа не возникает ошибки, а выдается по умолчанию значение None:

d.get("list")

d["list"]

**print**(d.get(3))

Это значение можно изменить, указав его вторым аргументом:

**print**(d.get(3, False) )

Похожий метод

dict.**setdefault**(key[, default]) возвращает значение, ассоциированное с ключом key и если его нет, то добавляет в словарь со значением None, либо default – если оно указано:

d.setdefault("list")

d.setdefault(3)

Добавит ключ 3 со значением None.

Удалим его:

**del** d[3]

d.setdefault(3, "three")

тогда добавится этот ключ со значением «three». То есть, этот метод способен создать новую запись, но только в том случае, если указанный ключ изначально отсутствовал в словаре.

Следующий метод удаляет указанный ключ и возвращает его значение. Если в нем указывается несуществующий ключ, то возникает ошибка: Но мы можем легко исправить ситуацию, если в качестве второго аргумента указать значение, возвращаемое при отсутствии ключа:

d.**pop**(3)

d.pop("abc")

d.pop("abc", False)

Здесь возвратится False. Если же ключ присутствует, то возвращается его значение.

Следующий метод

d.**popitem**()

выполняет удаление произвольной записи из словаря. Если словарь пуст, то возникает ошибка:

d2 = {}

d2.popitem()

Далее, метод

d.**keys**()

возвращает коллекцию ключей. По умолчанию цикл for обходит именно эту коллекцию, при указании словаря:

d = {True: 1, False: "Ложь", "list": [1,2,3], 5: 5}

**for** x **in** d:

**print**(x)

то есть, эта запись эквивалента такой:

**for** x **in** d.**keys**():

Если же вместо keys записать метод values:

**for** x **in** d.**values**():

то обход будет происходить по значениям, то есть, метод values возвращает коллекцию из значений словаря.

Последний подобный метод **items** возвращает записи в виде кортежей: ключ, значение, к элементу кортежа можно обратиться по индексу и вывести отдельно ключи и значения:

**for** key, value **in** d.items():

**print**(key, value)

**Примечание .**

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 .**

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.** О том, как устроен словарь (тип dict) в Python можно почитать в [статье](https://habr.com/ru/post/432996/). Исходный код словаря (тип dict) в Python можно найти [тут](https://github.com/python/cpython/blob/main/Objects/dictobject.c) Операция конкатенации + и умножения на число \* **недоступны** для словарей.

**Подвиг .**

Вводятся данные в формате ключ=значение в одну строчку через пробел. Необходимо на их основе создать словарь d, затем удалить из этого словаря ключи 'False' и '3', если они существуют. Ключами и значениями словаря являются строки. Вывести полученный словарь на экран

**Sample Input:**

лена=имя дон=река москва=город False=ложь 3=удовлетворительно True=истина

**Sample Output:**

('True', 'истина') ('дон', 'река') ('лена', 'имя') ('москва', 'город')

**Подвиг .** Вводятся номера телефонов в одну строчку через пробел с разными кодами стран: +7, +6, +2, +4 и т.д. Необходимо составить словарь d, где ключи - это коды +7, +6, +2 и т.п., а значения - список номеров (следующих в том же порядке, что и во входной строке) с соответствующими кодами. Полученный словарь вывести командой: print(\*sorted(d.items()))

**Sample Input:**

+71234567890 +71234567854 +61234576890 +52134567890 +21235777890 +21234567110 +71232267890

**Sample Output:**

('+2', ['+21235777890', '+21234567110'])

'+5', ['+52134567890'])

('+6', ['+61234576890']) ('+7', ['+71234567890', '+71234567854', '+71232267890'])