Словари

Рассмотрим еще один составной тип данных, называемый словарем, который похож на список в том, что он представляет собой набор объектов.

Вот что вы узнаете из этого урока: мы рассмотрим основные характеристики словарей Python и узнаем, как получить доступ к значениям словаря и управлять ими.
После того, как вы закончите этот урок, вы будете понимать, когда стоит использовать данный тип и как с ним работать.

Содержание

- Определение словаря
- Доступ к значениям словаря
- Ключи словаря и индексы списка
- Постепенное создание словаря
- Ограничения для ключей словаря
- Ограничения на значения словаря
- Операторы и встроенные функции

  Встроенные словарные методы
  - d.clear()
  - d.get([,])
  - d.items()
  - d.keys()
  - d.values()
  - d.pop([,])
  - d.popitem()
  - d.update()
  - Вывод

Словари и списки имеют следующие характеристики:

- Оба изменчивы.
- Оба динамичны. Они могут увеличиваться и уменьшаться по мере необходимости.
- Оба могут быть вложенными. Список может содержать другой список. Словарь может содержать другой словарь. Словарь также может содержать список, и наоборот.

Словари отличаются от списков в первую очередь способом доступа к элементам:

- Доступ к элементам списка осуществляется по их положению в списке через индексацию.
- Доступ к элементам словаря осуществляется с помощью ключей.

Определение словаря

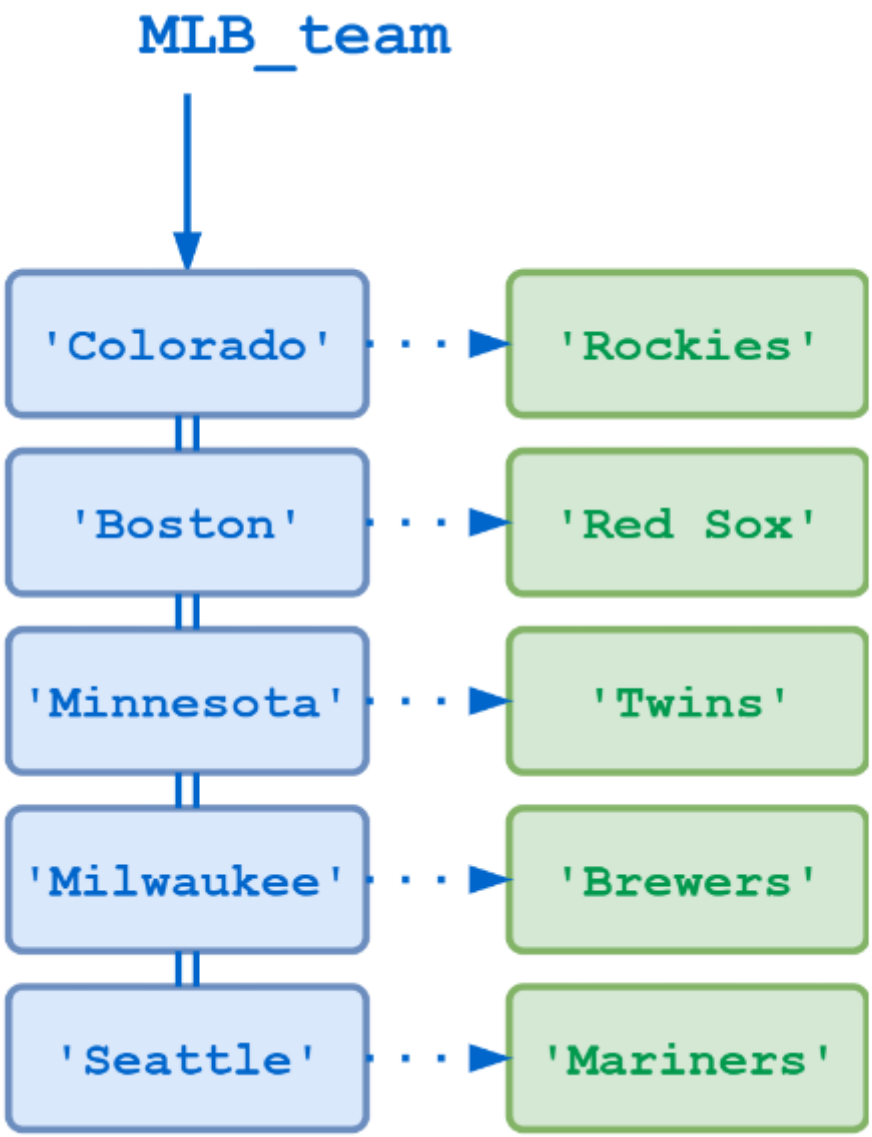
Словари — это реализация в Python структуры данных, более известной как ассоциативный массив. Словарь состоит из набора пар ключ-значение. Каждая пара «ключ-значение» сопоставляет ключ с соответствующим значением.

Вы можете определить словарь, заключив список пар ключ-значение, разделенных запятыми, в фигурные скобки ({}). Двоеточие (:) отделяет каждый ключ от связанного с ним значения:

```
#
d = {
        < key1 >: < value1 >,
        < key2 >: < value2 >,
        < key3 >: < value1 >,
                ...
        < keyn >: < valuen >,
}
```

В следующем примере ключем является название штата, а значением — название бейсбольной команды:

```
>>> MLB_team = {
        'Colorado'  : 'Rockies',
        'Boston'    : 'Red Sox',
        'Minnesota' : 'Twins',
        'Milwaukee' : 'Brewers',
        'Seattle'   : 'Mariners',
}
```



Сопоставление местоположения с командой в словаре MLB_team

Также можно создать словарь с помощью встроенной функции dict(). Аргумент dict() должен быть последовательностью пар ключ-значение. Для этого хорошо подходит список кортежей:

```
#
d = dict([
        (< key >, < value >),
        (< key >, < value >),
        .
        .
        .
        (< key >, < value >),
])
```

Словарь MLB_team можно определить так:

```
>>> MLB_team = dict([
    ('Colorado', 'Rockies'),
    ('Boston', 'Red Sox'),
    ('Minnesota', 'Twins'),
    ('Milwaukee', 'Brewers'),
    ('Seattle', 'Mariners')])
```

Если значения ключа являются простыми строками, их можно указать как аргументы ключевого слова. Итак, вот еще один способ определить MLB_team:

```
>>> MLB_team = dict(
    Colorado= 'Rockies',
    Boston= 'Red Sox',
    Minnesota= 'Twins',
    Milwaukee= 'Brewers',
    Seattle= 'Mariners')
```

После того, как вы определили словарь, вы можете отобразить его содержимое так же, как и для списка. Все три определения, показанные выше, при отображении выглядят следующим образом:

```
>>> type(MLB_team)

MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins', 'Milwaukee':
'Brewers', 'Seattle': 'Mariners'}
```

Записи в словаре отображаются в том порядке, в котором они были определены. Доступ к элементам словаря не осуществляется по числовому индексу:

```
>>> MLB_team[1]
Traceback (most recent call last):
  File "", line 1, in
    MLB_team[1]
KeyError: 1
```

Доступ к значениям словаря

Конечно, элементы словаря должны быть доступны. Как получить доступ к элементам словаря?
Необходимо указать ключ в квадратных скобках ([]):

```
>>> MLB_team['Minnesota']
'Twins'
>>> MLB_team['Colorado']
'Rockies'
```

Если вы ссылаетесь на ключ, которого нет в словаре, Python выдает ошибку:

```
>>> MLB_team['Toronto']
Traceback (most recent call last):
  File "", line 1, in
    MLB_team['Toronto']
KeyError: 'Toronto'
```

Для добавления записи в существующий словарь необходимо ввести новый ключ и значение:

```
>>> MLB_team['Kansas City'] = 'Royals'
>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins', 'Milwaukee':
'Brewers', 'Seattle': 'Mariners', 'Kansas City': 'Royals'}
```

Если вы хотите изменить запись, вы можете просто присвоить новое значение существующему ключу:

```
>>> MLB_team['Seattle'] = 'Seahawks'
>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins', 'Milwaukee':
'Brewers', 'Seattle': 'Seahawks', 'Kansas City': 'Royals'}
```

Чтобы удалить запись, используйте оператор удаления del, указав ключ для удаления:

```
>>> del MLB_team['Seattle']
>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins', 'Milwaukee':
'Brewers', 'Kansas City': 'Royals'}
```

Ключи словаря и индексы списка

Интерпретатор показывает ошибку (KeyError), когда к словарю обращаются с неопределенным ключом или по числовому индексу:

```
>>> MLB_team['Toronto']
Traceback (most recent call last):
  File "", line 1, in
    MLB_team['Toronto']
KeyError: 'Toronto'

>>> MLB_team[1]
Traceback (most recent call last):
  File "", line 1, in
    MLB_team[1]
KeyError: 1
```

По сути, это та же ошибка. В последнем случае [1] выглядит как числовой индекс, но это не так.
Объект любого неизменяемого типа может использоваться как ключ словаря. Соответственно, нет причин, по которым нельзя использовать целые числа:

```
>>> d = {0: 'a', 1: 'b', 2: 'c', 3: 'd'}
>>> d
{0: 'a', 1: 'b', 2: 'c', 3: 'd'}
>>> d[0]
'a'
>>> d[2]
'c'
```

В выражениях MLB_team[1], d[0] и d[2] числа в квадратных скобках выглядят как индексы. Но они не имеют ничего общего с порядком в словаре. Python интерпретирует их как ключи словаря. Если вы определите этот же словарь в обратном порядке, вы все равно получите те же значения, используя те же ключи:

```
>>> d = {3: 'd', 2: 'c', 1: 'b', 0: 'a'}
>>> d
{3: 'd', 2: 'c', 1: 'b', 0: 'a'}
>>> d[0]
'a'
>>> d[2]
'c'
```

Синтаксис может выглядеть похожим, но вы не можете рассматривать словарь как список:

```
>>> type(d)


>>> d[-1]
Traceback (most recent call last):
  File "", line 1, in
    d[-1]
KeyError: -1

>>> d[0:2]
Traceback (most recent call last):
  File "", line 1, in
    d[0:2]
TypeError: unhashable type: 'slice'

>>> d.append('e')
Traceback (most recent call last):
  File "", line 1, in
    d.append('e')
AttributeError: 'dict' object has no attribute 'append'
```

Примечание. Хотя доступ к элементам в словаре не зависит от порядка, Python гарантирует, что порядок элементов в словаре сохраняется. При отображении элементы будут отображаться в том порядке, в котором они были определены, и итерация по ключам также будет происходить в этом порядке. Элементы, добавленные в словарь, добавляются в конце. Если элементы удаляются, порядок остальных элементов сохраняется.
На такое сохранение порядка появилось совсем недавно. Оно было добавлено как часть спецификации языка Python в версии 3.7. Данный порядок сохранялся и в версии 3.6 случайно, без гарантированной спецификацией языка.

Постепенное создание словаря

Определение словаря с использованием фигурных скобок и списка пар ключ-значение, как показано выше, удобно, если вы заранее знаете все ключи и значения. Но что делать, если вы хотите создать словарь в процессе работы программы?
Вы можете начать с создания пустого словаря, который обозначен пустыми фигурными скобками. Затем вы можете добавлять новые ключи и значения по одному:

```
>>> person = {}
>>> type(person)


>>> person['fname'] = 'Joe'
>>> person['lname'] = 'Fonebone'
>>> person['age'] = 51
>>> person['spouse'] = 'Edna'
>>> person['children'] = ['Ralph', 'Betty', 'Joey']
>>> person['pets'] = {'dog':  'Fido', 'cat': 'Sox'}
```

После создания словаря таким образом доступ к его значениям осуществляется так же, как и к любому другому словарю:

```
>>> person
{'fname': 'Joe', 'lname': 'Fonebone', 'age': 51, 'spouse': 'Edna', 'children':
['Ralph', 'Betty', 'Joey'], 'pets': {'dog': 'Fido', 'cat': 'Sox'}}

>>> person['fname']
'Joe'
>>> person['age']
51
>>> person['children']
['Ralph', 'Betty', 'Joey']
```

Для получения значений в подсписке или подсловаре требуется дополнительный индекс или ключ:

```
>>> person['children']
['Ralph', 'Betty', 'Joey']
>>> person['children'][-1]
'Joey'
>>> person['pets']['cat']
'Sox'
```

В этом примере демонстрируется еще одна особенность словарей: значения, содержащиеся в словаре, не обязательно должны быть одного типа.
В person некоторые значения являются строками, одно — целое число, одно — список, а третье — другой словарь.
Ключи так же, как значения в словаре не обязательно должны быть одного и того же типа:

```
>>> foo = {42: 'aaa', 2.78: 'bbb', True: 'ccc'}
>>> foo
{42: 'aaa', 2.78: 'bbb', True: 'ccc'}
>>> foo[42]
'aaa'
>>> foo[2.78]
'bbb'
>>> foo[True]
'ccc'
```

Здесь один из ключей — целое число, один — число с плавающей запятой, а третий — логическое.
Обратите внимание, насколько универсальны словари Python. В MLB_team информация о названии бейсбольной команды сохраняется для каждого штата. Словарь person хранит различные типы данных для одного человека.
Вы можете использовать словари для самых разных целей, потому что существует очень мало ограничений на разрешенные ключи и значения. Но они существуют. Читайте дальше!

Ограничения для ключей словаря

Почти любой тип значения может использоваться в качестве словарного ключа в Python. Вы только что видели пример, где в качестве ключей используются целочисленные, плавающие и логические объекты:

```
>>> foo = {42: 'aaa', 2.78: 'bbb', True: 'ccc'}
>>> foo
{42: 'aaa', 2.78: 'bbb', True: 'ccc'}
```

Вы даже можете использовать встроенные объекты, такие как типы и функции:

```
>>> d = {int: 1, float: 2, bool: 3}
>>> d
{: 1, : 2, : 3}
>>> d[float]
2

>>> d = {bin: 1, hex: 2, oct: 3}
>>> d[oct]
3
```

Однако есть пара ограничений, которым должны соответствовать словарные ключи.

Во-первых, данный ключ может появиться в словаре только один раз. Повторяющиеся ключи не допускаются. Словарь сопоставляет каждый ключ с соответствующим значением, поэтому нет смысла отображать конкретный ключ

более одного раза.

Вы видели выше, что когда вы присваиваете значение уже существующему ключу словаря, он не добавляет ключ второй раз, а заменяет существующее значение:

```
>>> MLB_team = {
        'Colorado'  : 'Rockies',
        'Boston'    : 'Red Sox',
        'Minnesota' : 'Twins',
        'Milwaukee' : 'Brewers',
        'Seattle'   : 'Mariners',
}
>>> MLB_team['Minnesota']='Timberwolves'
>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Timberwolves',
'Milwaukee': 'Brewers', 'Seattle': 'Mariners'}
```

Точно так же, если вы один и тот же ключ укажете два раза, то второе значение заменит первое:

```
>>> MLB_team = {
        'Colorado'  : 'Rockies',
        'Boston'    : 'Red Sox',
        'Minnesota' : 'Twins',
        'Milwaukee' : 'Brewers',
        'Seattle'   : 'Mariners',
        'Minnesota' : 'Twins'
}
>>> MLB_team
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins', 'Milwaukee':
'Brewers', 'Seattle': 'Mariners'}
```

Во-вторых, ключ словаря должен иметь неизменяемый тип. Вы уже видели примеры, в которых несколько неизменяемых типов — integer, float, string и Boolean — служили ключами словаря.

Кортеж также может быть ключом словаря, потому что кортежи неизменяемы:

```
>>> d={(1,1):'a',(1,2):'b',(2,1):'c',(2,2):'d'}
>>> d[(1,1)]
'a'
>>> d[(2,1)]
'c'
```

(Вспомните из обсуждения кортежей, что одно из оснований для использования кортежа вместо списка состоит в том, что существуют обстоятельства, когда требуется неизменяемый тип. Это одно из них.)

Однако ни список, ни другой словарь не могут служить ключом словаря, потому что списки и словари изменяемы:

```
>>> d={[1,1]:'a',[1,2]:'b',[2,1]:'c',[2,2]:'d'}
Traceback (most recent call last):
  File "", line 1, in
    d={[1,1]:'a',[1,2]:'b',[2,1]:'c',[2,2]:'d'}
TypeError: unhashable type: 'list'
```

Примечание. Почему в сообщении об ошибке написано «нехэшируемое»?

Технически не совсем правильно говорить, что объект должен быть неизменным, чтобы его можно было использовать в качестве словарного ключа. Точнее, объект должен быть хешируемым, что означает, что его можно передать хеш-функции. Хеш-функция принимает данные произвольного размера и сопоставляет их с относительно более простым значением фиксированного размера, называемым хеш-значением(или просто хешем), которое используется для поиска и сравнения в таблице.

Встроенная hash() функция Python возвращает хеш-значение для хешируемого объекта и вызывает исключение для объекта, который таким не является:

```
>>> hash('foo')
-1366538220

>>> hash([1,2,3])
Traceback (most recent call last):
  File "", line 1, in
    hash([1,2,3])
TypeError: unhashable type: 'list'
```

Все встроенные неизменяемые типы, о которых вы узнали до сих пор, являются хешируемыми, а изменяемые типы(списки и словари) — нет. Можно считать, что хэшируемые и неизменяемые объекты — это синонимы.

Однако, в будущих уроках вы встретите изменяемые объекты, которые также могут быть хешированы.

Ограничения на значения словаря

Напротив, нет ограничений на значения словаря. Совсем нет. Значение словаря может быть любым типом объекта, поддерживаемым Python, включая изменяемые типы, такие как списки и словари, а также определяемые пользователем объекты, о которых вы узнаете в следующих уроках.

Также нет ограничений на то, чтобы определенное значение появлялось в словаре несколько раз:

```
>>> d={0:'a',1:'a',2:'a',3:'a'}
>>> d
{0: 'a', 1: 'a', 2: 'a', 3: 'a'}
>>> d[0]==d[1]==d[2]
True
```

Операторы и встроенные функции

Вы уже познакомились со многими операторами и встроенными функциями, которые можно использовать со строками, списками и кортежами. Некоторые из них также работают со словарями.

Например, in и not in операторы возвращают True или False в соответствии с тем, подходит ли указанный операнд в качестве ключа в словаре:

```
MLB_team = {
        'Colorado'  : 'Rockies',
        'Boston'    : 'Red Sox',
        'Minnesota' : 'Twins',
        'Milwaukee' : 'Brewers',
        'Seattle'   : 'Mariners',
        'Minnesota' : 'Twins'
}
>>> 'Milwaukee' in MLB_team
True
>>> 'Toronto' in MLB_team
False
>>> 'Toronto' not in MLB_team
True
```

Можно использовать оператор in вместе с оценкой True или False, чтобы избежать появления ошибки при попытке доступа к ключу, которого нет в словаре:

```
>>> MLB_team['Toronto']
Traceback (most recent call last):
  File "", line 1, in
    MLB_team['Toronto']
KeyError: 'Toronto'

>>> 'Toronto' in MLB_team and MLB_team['Toronto']
False
```

Во втором случае из-за оценки False выражение MLB_team[‘Toronto’] не оценивается, поэтому ошибка не возникает.

Функция len() возвращает количество пар ключ-значение в словаре:

```
>>> MLB_team = {
        'Colorado'  : 'Rockies',
        'Boston'    : 'Red Sox',
        'Minnesota' : 'Twins',
        'Milwaukee' : 'Brewers',
        'Seattle'   : 'Mariners',
        'Minnesota' : 'Twins'
}
>>> len(MLB_team)
5
```

Встроенные словарные методы

Как и в случае со строками и списками, есть несколько встроенных методов, которые можно использовать в словарях. Фактически, в некоторых случаях методы списка и словаря имеют одно и то же имя (при обсуждении объектно-ориентированного программирования вы увидите, что для разных типов вполне приемлемо использовать функции с одинаковыми именами.)

Ниже приводится обзор методов, применимых к словарям:

```
>>> d.clear()
# Очищает словарь
```

d.clear() очищает словарь для всех пар ключ-значение:

```
>>> d={'a':10,'b':20,'c':30}
>>> d
{'a': 10, 'b': 20, 'c': 30}
>>> d.clear()
>>> d
{}
```

```
d.get(<key>[,<default>])
# Возвращает значение ключа, если он существует в словаре
```

Метод d.get() предоставляет удобный способ получения значения ключа из словаря без предварительной проверки наличия ключа и без возникновения ошибки.

d.get() ищет значение по ключу и возвращает его, если оно найден. Если не найдено, возвращается None:

```
>>> d={'a':10,'b':20,'c':30}
>>> print(d.get('b'))
20
>>> print(d.get('z'))
None
```

Если указано необязательный аргумент и значение не найдено, то возвращается необязательный элемент:

```
>>> print(d.get('z',-1))
-1
```

```
d.items()
# Возвращает список пар ключ-значение в словаре
```

d.items() возвращает список кортежей, содержащих пары ключ-значение в d. Первый элемент в каждом кортеже — это ключ, а второй элемент — значение ключа:

```
>>> d={'a':10,'b':20,'c':30}
>>> d
{'a': 10, 'b': 20, 'c': 30}

>>> list(d.items())
[('a', 10), ('b', 20), ('c', 30)]
>>> list(d.items())[1][0]
'b'
>>> list(d.items())[1][1]
20
```

```
d.keys()
# Возвращает список ключей в словаре
```

d.keys() возвращает список всех ключей в d:

```
>>> d = {'a': 10, 'b' : 20, 'c' : 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}
>>> list(d.keys())
['a', 'b', 'c']
```

```
d.values()
# Возвращает список значений в словаре
```

d.values() возвращает список всех значений в d:

```
>>> d = {'a': 10, 'b' : 20, 'c' : 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}
>>> list(d.values())
[10, 20, 30]
```

Любые повторяющиеся значения d будут возвращаться столько раз, сколько они встречаются:

```
>>> d = {'a': 10, 'b' : 10, 'c' : 10}
>>> d
{'a': 10, 'b': 10, 'c': 10}
>>> list(d.values())
[10, 10, 10]
```

Техническое примечание: d.items(), d.keys() и d.values() это методы, которые фактически возвращают вид объекта. Для практических целей вы можете рассматривать эти методы как возвращающие списки ключей и значений словаря.

```
d.pop(< key >[,< default >])
# Удаляет ключ из словаря, если он присутствует, и возвращает его значение.
```

Если ключ присутствует в d, то d.pop() удаляет его и возвращает связанное с ним значение:

```
>>> d = {'a': 10, 'b' : 20, 'c' : 30}
>>> d.pop('b')
20
>>> d
{'a': 10, 'c': 30}
```

d.pop() показывает ошибку, если указанный ключ не находится в d:

```
>>> d = {'a': 10, 'b' : 20, 'c' : 30}
>>> d.pop('z')
Traceback (most recent call last):
  File "", line 1, in
    d.pop('z')
KeyError: 'z'
```

Если отсутствует d и указан необязательный аргумент, то возвращается это значение, и ошибка не возникает:

```
>>> d = {'a': 10, 'b' : 20, 'c' : 30}
>>> d.pop('z', -1)
-1
>>> d
{'a': 10, 'b': 20, 'c': 30}
```

```
d.popitem()
# Удаляет пару ключ-значение из словаря.
```

d.popitem() удаляет последнюю добавленную пару ключ-значение d и возвращает ее как кортеж:

```
{'a': 10, 'b': 20, 'c': 30}
>>> d = {'a': 10, 'b' : 20, 'c' : 30}
>>> d.popitem()
('c', 30)
>>> d
{'a': 10, 'b': 20}
>>> d.popitem()
('b', 20)
>>> d
{'a': 10}
```

Если d пусто, d.popitem() возникает ошибка:

```
>>> d={}
>>> d.popitem()
Traceback (most recent call last):
  File "", line 1, in
    d.popitem()
KeyError: 'popitem(): dictionary is empty'
```

Примечание. В версиях Python до 3.6 popitem() возвращала бы произвольную (случайную) пару ключ-значение, поскольку словари Python были неупорядоченными.

```
d.update()
# Объединяет словарь с другим словарем или с парами ключ-значение
```

Если это словари, d.update() объединяет записи из этих словарей. Для каждого ключа в:

- Если ключ отсутствует d, пара ключ-значение добавляется в d.
- Если ключ уже присутствует в d, соответствующее значение в d для этого ключа обновляется.

Вот пример объединения двух словарей:

```
>>> d1={'a' : 10, 'b' : 20, 'c' : 30}
>>> d2={'b' : 200, 'd' : 400}
>>> d1.update(d2)
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

В этом примере ключ ‘b’ уже существует в d1, поэтому его значение обновляется до значения для этого ключа из d2(200). Однако, отсутствует ключ ‘d’ в d1, так что пара ключ-значение (‘d’:400) добавляется из d2.

Также может добавляться в словарь последовательностью пар ключ-значение, аналогично тому, как dict() функция используется для определения словаря. Например, можно указать список кортежей:

```
>>> d1={'a' : 10, 'b' : 20, 'c' : 30}
>>> d1.update([('b', 200), ('d', 400)])
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

Или значения для объединения можно указать в виде списка аргументов ключевого слова:

```
>>> d1={'a' : 10, 'b' : 20, 'c' : 30}
>>> d1.update(b=200,d=400)
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

Вывод

В этом уроке вы рассмотрели основные свойства словаря Python и узнали, как получить доступ и управлять данными словаря.

Списки и словари — два наиболее часто используемых типа Python. Как вы могли заметить, они имеют несколько общих черт, но отличаются способом доступа к элементам. Доступ к элементам списков осуществляется по числовому индексу в зависимости от порядка, а к элементам словаря — по ключу.

Из-за этой разницы списки и словари подходят для разных ситуаций. Теперь вы должны понять, что лучше всего подходит для конкретной ситуации.
Далее вы узнаете о наборах Python. Набор — это еще один составной тип данных, но он сильно отличается от списка или словаря.

Оригинал: Dictionaries in Python
GitHub