Списки, кортежи и словари в Python

Python содержит важные типы данных, которыми вы с высокой вероятностью будете использовать каждый день. Они называются **списки**, **кортежи** и **словари**. Цель данной статьи познакомить вас с ними поближе. Они не очень сложные, так что надеюсь, вы научитесь, как использовать их по назначению. После освоения этих трех типов данных, в сочетании со строками из предыдущей статьи, вы ощутимо продвинетесь в изучении Python. Вам понадобится каждый из этих четырех кирпичиков для создания 99% приложений.

Списки

Списки Python схожи с **массивами** в других языках. В Python, пустой список может быть создан следующим образом:

```python
my_list = []
my_list = list()
```

Как вы видите, вы можете **создать список** при помощи квадратных скобок, или при помощи встроенного инструмента Python – **list**. Список состоит из таких элементов, как строки, цифры, объекты и смеси типов. Давайте взглянем на несколько примеров:

```python
my_list = [1, 2, 3]
my_list2 = ["a", "b", "c"]
my_list3 = ["a", 1, "Python", 5]
```

Первый список содержит 3 числа, второй 3 строки, третий содержит смесь. Вы также можете создавать списки списков, вот так:

```python
my_nested_list = [my_list, my_list2]
print(my_nested_list) # [[1, 2, 3], ['a', 'b', 'c']]
```

В какой-то момент вам может понадобиться **скомбинировать два списка** вместе. Первый способ сделать это – при помощи метода **extend**:

```python
combo_list = [1]
one_list = [4, 5]
a = combo_list.extend(one_list)

print(a) # [1, 4, 5]
```

Немного проще будет просто **добавить два списка вместе**.

```python
my_list = [1, 2, 3]
my_list2 = ["a", "b", "c"]

combo_list = my_list + my_list2
print(combo_list) # [1, 2, 3, 'a', 'b', 'c']
```

Да, это именно настолько просто. Вы также можете **сортировать список**. Давайте уделим немного времени и взглянем на то, как это делается:

```python
alpha_list = [34, 23, 67, 100, 88, 2]
alpha_list.sort()

print(alpha_list) # [2, 23, 34, 67, 88, 100]
```

Получилось. Видите? Давайте взглянем на еще один пример, чтобы закрепить результат:

```python
alpha_list = [34, 23, 67, 100, 88, 2]
sorted_list = alpha_list.sort()

```

```
print(sorted_list) # None
```

В этом примере мы попытались назначить сортированный список переменной. Однако, когда вы вызываете метод **sort**() в списке, он сортирует список на месте. Так что если вы попробуете назначить результат другой переменной, тогда возникнет объект **None**, который аналогичен объекту **Null** в других языках. Таким образом, когда вам нужно отсортировать что-нибудь, просто помните, что вы сортируете на месте, и вы не можете назначить объект другой переменной.
Вы можете разрезать список также, как вы делаете это со строкой:

Python

```
a = alpha_list[0:3]
print(a) # [2, 23, 34]
```

Данный код выдает список из трех первых элементов.

Кортежи

Кортеж похож на список, но вы создаете его с **круглыми скобками**, вместо квадратных. Вы также можете использовать встроенный инструмент для создания кортежей. Разница в том, что **кортеж неизменный**, в то время как **список** может меняться. Давайте взглянем на несколько примеров:

Python

```
my_tuple = (1, 2, 3, 4, 5)
a = my_tuple[0:3]
print(a) # (1, 2, 3)

another_tuple = tuple()
abc = tuple([1, 2, 3])
```

Данный код демонстрирует способ **создания кортежа** с пятью элементами. Также он говорит нам о том, что мы можете делать нарезку кортежей. Однако, вы не можете **сортировать кортеж**! Последние два примера показывают, как создавать кортеж при помощи ключевого слова **tuple** (которое и переводится как «кортеж»). Первый код просто создает пустой кортеж, в то время как во втором примере кортеж содержит три элемента. Обратите внимание на то, что в нем есть список. Это пример конвертации. Мы можем менять или конвертировать объект из одного типа данных в другой. В нашем случае, мы конвертируем **список в кортеж**. Если вы хотите превратить кортеж **abc** обратно в список, вы можете сделать это следующим образом:

Python

```
abc_list = list(abc)
```

Для повторения, данный код конвертирует кортеж в список при помощи функции **list**.



Словари

Словарь Python, по большей части, представляет собой **хэш-таблицу**. В некоторых языках, словари могут упоминаться как ассоциативная память, или **ассоциативные массивы**. Они индексируются при помощи ключей, которые могут быть любого неизменяемого типа. Например, строка или число могут быть ключом. Вам обязательно стоит запомнить тот факт, что словарь – это неупорядоченный набор пар *ключ:значение*, и ключи обязательно должны быть уникальными.

Вы можете получить список ключей путем вызова метода **keys**() в том или ином словаря. Чтобы проверить, присутствует ли ключ в словаре, вы можете использовать ключ **in** в Python. В некоторых старых версиях Python (с 2.3 и более ранних, если быть точным), вы увидите ключевое слово **has_key**, которое используется для проверки наличия ключа в словаре. Данный ключ является устаревшим в Python 2.X, и был удален, начиная с версии 3.X. Давайте попробуем создать наш первый словарь:

Python

```
my_dict = {}
another_dict = dict()

my_other_dict = {"one":1, "two":2, "three":3}
print(my_other_dict) # {'three': 3, 'two': 2, 'one': 1}
```

Первые два примера показывают, как **создавать пустой словарь**. Все словари находятся в **фигурных скобках**. Последняя строчка показывает, что мы имеем в виду, когда говорим «неупорядоченный словарь». Теперь настало время узнать, как принимаются значения в словаре.

Python

```python
my_other_dict = {"one":1, "two":2, "three":3}

print(my_other_dict["one"]) # 1

my_dict = {"name":"Mike", "address":"123 Happy Way"}
print(my_dict["name"]) # 'Mike'
```

В первом примере, мы использовали словарь из предыдущего примере, и вытащили значение, связанное с ключом под названием **one**. Второй пример демонстрирует, как задавать значение ключу **name**. Теперь попробуем узнать, находится ли ключ в словаре или нет:

Python

```python
print("name" in my_dict) # True

print("state" in my_dict) # False
```

Что-ж, если ключ в словаре, Python выдает нам Boolean True. В противном случае, мы получаем **Boolean False**. Если вам нужно получить список ключей в словаре, вам нужно сделать следующее:

Python

```python
print(my_dict.keys()) # dict_keys(['name', 'address'])
```

В Python 2, метод keys дает нам список. Но в Python 3 он дает объект **view**. Это дает разработчику возможность **обновлять словарь**, так что view также обновится. Обратите внимание на то, что когда мы используем ключевое слово **in** для текста содержимого словаря, лучше будет сделать это в словаре, а не в списке, выдаваемом методом keys. Смотрим ниже:

Python

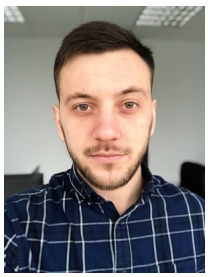
```python
if "name" in my_dict # Такая конструкция правильная

if "name" in my_dict.keys() # Работает но медленее
```

Пока это, возможно, не говорит вам о многом, во время реальной работы ситуация будет другая и каждая секунда будет важна. При создании тысячи файлов для обработки, эти маленькие хитрости могут уберечь вас от бесполезной траты времени.

Подведем итоги

В данной статье мы узнали, как **создавать списки**, **кортежи** и **словари** в Python. Убедитесь в том, что все, что изложено в данной статье понятно для вас, перед тем как двигаться дальше. Эти основы необходимо знать при разработке любого приложения. Вы будете создавать сложные структуры данных, пользуясь этими тремя кирпичиками каждый день, если планируете быть программистом Python, разумеется. Каждый из этих типов данных может быть размещен в другой. Например, вы можете создавать словарь кортежей, или кортеж, созданный из нескольких словарей, и так далее.



Vasile Buldumac

Являюсь администратором нескольких порталов по обучению языков программирования Python, Golang и Kotlin. В составе небольшой команды единомышленников, мы занимаемся популяризацией языков программирования на русскоязычную аудиторию. Большая часть статей была адаптирована нами на русский язык и распространяется бесплатно.

E-mail: vasile.buldumac@ati.utm.md

Образование
Universitatea Tehnică a Moldovei (*utm.md*)

2014 — 2018 Технический Университет Молдовы, ИТ-Инженер. Тема дипломной работы «*Автоматизация покупки и продажи криптовалюты используя технический анализ*»

2018 — 2020 Технический Университет Молдовы, Магистр, Магистерская диссертация «*Идентификация человека в киберпространстве по фотографии лица*»