В ваших программах часто требуется группировать несколько элементов, чтобы обрабатывать их как единый объект. Для этого вам нужно будет использовать разные коллекции. Одним из наиболее важных и полезных наборов в Python является **список**.

**Создание и печать списков**

Посмотрите на простой список, в котором хранятся несколько названий пород собак:

dog\_breeds = ['corgi', 'labrador', 'poodle', 'jack russell']

print(dog\_breeds) # ['corgi', 'labrador', 'poodle', 'jack russell']

В первой строке мы используем квадратные скобки, чтобы создать список, содержащий четыре элемента, а затем присвоить его dog\_breeds переменной. Во второй строке список печатается через имя переменной. Все элементы печатаются в том же порядке, в каком они были сохранены в списке, потому что списки **упорядочены**.

Вот еще один список, содержащий пять целых чисел:

numbers = [1, 2, 3, 4, 5]

print(numbers) # [1, 2, 3, 4, 5]

Другой способ создать список - это вызвать list функцию. Он используется для создания списка из **итерируемого** объекта: то есть такого объекта, из которого вы можете получать его элементы один за другим. Концепция повторяемости будет подробно объяснена далее, но давайте посмотрим на примеры ниже:

list\_out\_of\_string = list('danger!')

print(list\_out\_of\_string) # ['d', 'a', 'n', 'g', 'e', 'r', '!']

list\_out\_of\_integer = list(235) # TypeError: 'int' object is not iterable

Таким образом, list функция создает список, содержащий каждый элемент из данного итеративного объекта. На данный момент помните, что **строка**является примером **итерируемого**объекта, а **целое**число является примером **не итерируемого** объекта. **Список**сам по себе также является **итерируемым**объектом.

Давайте также отметим разницу между list функцией и созданием списка с использованием квадратных скобок:

multi\_element\_list = list('danger!')

print(multi\_element\_list) # ['d', 'a', 'n', 'g', 'e', 'r', '!']

single\_element\_list = ['danger!']

print(single\_element\_list) # ['danger!']

Квадратные скобки и list функция также могут быть использованы для создания **пустых списков**, в которых вообще нет элементов.

empty\_list\_1 = list()

empty\_list\_2 = []

В следующих разделах мы рассмотрим, как заполнить пустые списки.

**Особенности списков**

Списки могутхранить **повторяющиеся значения** столько раз, сколько необходимо.

on\_off\_list = ['on', 'off', 'on', 'off', 'on']

print(on\_off\_list) # ['on', 'off', 'on', 'off', 'on']

Еще одна важная особенность списков заключается в том, что они могут содержать **различные типы** элементов, включая другие списки. Таким образом, нет ни ограничений, ни фиксированных типов списков, и вы можете добавить в свой список любые данные, которые вы хотите, как в следующем примере:

different\_objects = ['a', 1, 'b', 2, [1, 2, 3]]

**Длина списка**

Иногда вам нужно знать, сколько элементов в списке. Существует встроенная функция, вызываемая len, которая может быть применена к любому**итерируемому** объекту, и она возвращает просто **длину** этого объекта.

Таким образом, при применении к списку он возвращает количество элементов в этом списке.

numbers = [1, 2, 3, 4, 5]

print(len(numbers)) # 5

empty\_list = list()

print(len(empty\_list)) # 0

single\_element\_list = ['danger!']

print(len(single\_element\_list)) # 1

multi\_elements\_list = list('danger!')

print(len(multi\_elements\_list)) # 7

В приведенном выше примере вы можете увидеть, как работает len() функция. Опять же, обратите внимание на разницу между list() и [] применительно к строкам: это может привести не к тому, что вы ожидали.

**Краткие сведения**

В качестве резюме мы отмечаем, что списки являются:

* **упорядоченный**, то есть каждый элемент имеет фиксированную позицию в списке;
* **итеративный**, то есть вы можете получать их элементы один за другим;
* возможность хранить **повторяющиеся значения**;
* возможность хранить **различные типы элементов;**
* помимо использования для создания пустого списка,list() функция также может использоваться для создания списка из итерируемого объекта.

**Практика**



