# Урок 16.01. Кортежи.

Мы продолжаем изучать новые коллекции. И сегодня мы познакомимся с новым типом данных, который последний по счету, но не по значению коллекция в python. Которая используется также довольно часто.

И для этого, как всегда, рассмотрим такой код

def add\_num(seq, number):

 for i\_num in range(len(seq)):

 seq[i\_num] += number

origin\_list = [3, 1, 4, 1, 5]

add\_num(origin\_list, 5)

print(origin\_list)

В этой программе формируется список чисел, а потом к каждому числу этого списка в функции прибавляется заданное значение. В нашем случае это число 5, и этот список выводится на экран. Особенностью данной программы является то, что в функции не происходит возвращение списка (нет оператора return). Это возможно благодаря тому, что в функцию мы передаем не сам список на ссылку на адрес где лежит этот список (такой фокус происходит только для списков и подобных типов данных, но не проходит для простых типов данных, например, числовых и текстовых).

Но что делать, если нам нужно изменить список внутри функции и при этом не изменить его в основной программе. Такую задачу можно решить двумя способами

Первый способ: взять копию списка с помощью среза.

def add\_num(seq, number):

 for i\_num in range(len(seq)):

 seq[i\_num] += number

origin\_list = [3, 1, 4, 1, 5]

changed\_list = origin\_list[:]

add\_num(origin\_list, 5)

print(origin\_list)

print(changed\_list)

Такой способ работает. Однако у этого способа есть один недостаток, нам нужно каждый раз делать копию списка, и перед каждым вызовом функции сохранять эту копию и в случае необходимости восстанавливать. Поэтому нам подойдет второй способ.

Второй способ: изменим наш список на кортеж (заменим квадратные скобки на круглые). И теперь нашу последовательность чисел изменять нельзя, так как кортеж это такой список, который изменять нельзя. Кортеж по-английски будет *tuple*, и заменим в названии *list* на *tuple*.

def add\_num(seq, number):

 for i\_num in range(len(seq)):

 seq[i\_num] += number

origin\_tuple = (3, 1, 4, 1, 5)

changed\_tuple = origin\_tuple[:]

add\_num(origin\_tuple, 5)

print(origin\_tuple)

print(changed\_tuple)

Запустим и получим ошибку. Конечно, всё дело в том, что кортеж нельзя изменить, что мы и сделали. Так как нам это разрешить. Всё просто: нужно в начале функции преобразовать наш кортеж в список и вернуть этот измененный список.

def add\_num(seq, number):

 seq = list(seq)

 for i\_num in range(len(seq)):

 seq[i\_num] += number

 return seq

origin\_tuple = (3, 1, 4, 1, 5)

changed\_tuple = add\_num(origin\_tuple, 5)

print(origin\_tuple)

print(changed\_tuple)

Вот теперь всё работает. Таким образом, мы обеспечили защиту нашим данным.

Однако, на этом плюсы кортежей не заканчиваются, к ним еще относят:

а) уменьшение места для хранения данных;

б) прирост производительности при работе с кортежами.

Давайте изучим кортежи немного. Посмотрим доступные методы.

Посмотрим такую ситуацию:

list\_1 = [1, 1, 1]

tuple\_1 = tuple(list\_1)

print(list\_1)

print(tuple\_1)

tuple\_2 = (1, 1, list\_1)

print(tuple\_2)

tuple\_2[2][0] = 100

print(tuple\_2)

Как мы видим, в предпоследней строке мы вроде бы меняем значение в кортеже, однако это не так, потому что в кортеже хранится не сам список, а его адрес, и тогда мы меняем не кортеж на значение в списке, а в кортеже всё также остается адрес списка.

Хорошо. Давайте дальеш.

Например, нам нужно задать в одну переменную данные юзера: фамилию, имя, возраст. Так и сделаем

user = 'Petrov', 'Vova', 25

print(user)

И Python нам сам превратит эти три значения в кортеж, и даже не надо скобок круглых. А если нам нужно развернуть наш кортеж на три переменных, то сделать это также легко

user = 'Petrov', 'Vova', 25

print(user)

surname, name, age = user

print(surname, name, age)

И тут мы подходим к пониманию, где эти кортежи мы могли использовать не явно. Например, если нам нужно вернуть несколько переменных из функции

def get\_user():

 user\_name = 'Bob'

 user\_surname = 'Smith'

 user\_age = 15

 return user\_surname, user\_name, user\_age

surname\_2, name\_2, age\_2 = get\_user()

print(surname\_2, name\_2, age\_2)

Таким образом, мы можем возвращать несколько значений из функции, если еще не пробовали.