**Тема занятия №2: Числа, строки, списки, кортежи, множества, диапазоны, словари**

**2.1 Строки в Python**

Строка в Python – это последовательный набор символов, который может состоять как из цифр, так и из букв, и разделителей. Для создания строки мы используем кавычки (одинарные, двойные или тройные – не имеет значения). Строка считывается функцией input(). Например, 'Hello, world!', '1', '123', ' '.

В Python нет отдельного символьного типа – типа данных, объектами которого являются одиночные символы. Аналогом символа можно считать строку, состоящую из одного элемента, например, '1'.

Чтобы узнать длину строки существует функция len(). Например, s = '123' print(len(s))

Строки можно дублировать (умножать на число) и складывать (дописывать к одной строке другую – **конкатенация**). Например,

s1 = 'Hello'

s2 = 'World'

print(s1 \* s2)

print(s1 + s2)

**2.2 Срезы**

Срез (slice) – извлечение из строки одного символа или некоторого фрагмента строки (последовательности символов).

Срез из одного символа

Из строки можно извлечь один символ.

Для этого используется конструкция: s[i] – это срез, состоящий из одного символа, который имеет номер i. При этом считается, что нумерация начинается с числа 0.

Например, если s = 'Hello', то s[0] == 'H', s[1] == 'e', s[2] == 'l', s[3] == 'l', s[4] == 'o'. i – номер элемента – называется индексом.

Если указать отрицательное значение индекса, то номер будет отсчитываться с конца, начиная с номера -1.

Например, s[-1] == 'o', s[-2] == 'l', s[-3] == 'l', s[-4] == 'e', s[-5] == 'H'.

Таким образом, чтобы получить определенный символ из строки, нужно обратиться к нему по положительному или отрицательному номеру (индексу):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Строка s | H | e | l | l | o |
| Положительный индекс | s[0] | s[1] | s[2] | s[3] | s[4] |
| Отрицательный индекс | s[-5] | s[-4] | s[-3] | s[-2] | s[-1] |

**2.3 Подстрока**

Можно извлекать последовательность символов – подстроку. Для этого используется конструкция s[a, b], где a – индекс, с которого начинается нужная подстрока, а b – индекс, которым она заканчивается, при этом b не включен в диапазон.

Например, извлечем из строки 'Hello' подстроку 'Hell':

**s = 'Hello'**

**print(s[0:4])**

Или можно записать иначе, используя отрицательный индекс:

**s = 'Hello'**

**print(s[0:-1])**

Если срез начинается с 0, параметр a = 0 можно опустить. Тогда код выше будет выглядеть следующим образом:

**s = 'Hello'**

**print(s[:4])**

Если нужно взять срез с первого элемента и до конца, можно это записать так:

**s = 'Hello'**

**print(s[1:5])**

Если срез идет до конца строки, параметр b можно опустить. Например,

**s = 'Hello'**

**print(s[1:])**

Срез с шагом

Последний параметр среза – шаг. Аналогично функции range():

**s[a:b:d].**

Например, код, выполняющий извлечение из строки каждого второго символа, может выглядеть так:

**s = '123456789'**

**print(s[0:10:2])**

Пример выше можно записать проще, опустив параметры a и b, так как здесь они имеют значения начала и конца строки, совпадающие со стандартными:

**s = '123456789'**

**print(s[::2])**

Чтобы перевернуть строку, нужно использовать отрицательное значение шага. Например,

**s = '123456789'**

**print(s[::-1])**

Любые операции среза со строкой создают новые строки и никогда не меняют исходную строку. В Python строки являются неизменяемыми. Можно присвоить срез новой строке. Например,

**s = '123456789'**

**s1 = s[::-1]**

**print(s, s1)**

**2.4 Методы строк**

**Метод find()**

Метод find() находит в строке указанную подстроку. Функция возвращает индекс первого вхождения искомой подстроки. Если же подстрока не найдена, то метод возвращает значение -1. Например,

**s = 'Hello'**

**print(s.find('e'))**

**print(s.find('ll'))**

**print(s.find('L'))**

Если вызвать метод find() с тремя параметрами s.find(T, a, b), то поиск будет осуществляться в срезе s[a:b].

Например, чтобы найти если букву l в срезе s[:3] можно воспользоваться следующим кодом:

**s = 'Hello'**

**print(s.find('l', 0, 3)) # вернёт 2**

Если указать только два параметра s.find(T, a), то поиск будет осуществляться в срезе s[a:]. Код, который ищет символ e в срезе s[2:] может выглядеть так:

**s = 'Hello'**

**print(s.find('e', 2))**

Метод replace()

Метод replace() заменяет все вхождения одной строки на другую. Формат: s.replace(old, new) – заменить в строке s все подстроки old на подстроку new. Например,

**s = 'Hello'**

**print(s.replace('l', 'L'))**

Если методу replace задать еще один параметр: s.replace(old, new, count), то заменены будут не все вхождения, а только не больше, чем первые count из них. Например, чтобы заменить в строке 'AbrAkAdabra' только первые две строчные буквы a можно использовать следующий код:

**print('Abrakadabra'.replace('a', 'A', 2))**

**Метод count()**

Метод count() подсчитывает, сколько раз в строке встретилась другая строка. Например, print('Abracadabra'.count('a'))

При указании трех параметров s.count(T, a, b), будет выполнен подсчет числа вхождений строки T в срез s[a:b].

Рассмотрим основные методы строк, которые мы можем применить в приложениях:

isalpha(): возвращает True, если строка состоит только из алфавитных символов

islower(): возвращает True, если строка состоит только из символов в нижнем регистре

isupper(): возвращает True, если все символы строки в верхнем регистре

isdigit(): возвращает True, если все символы строки - цифры

isnumeric(): возвращает True, если строка представляет собой число

startswith(str): возвращает True, если строка начинается с подстроки str

endswith(str): возвращает True, если строка заканчивается на подстроку str

lower(): переводит строку в нижний регистр

upper(): переводит строку в вехний регистр

title(): начальные символы всех слов в строке переводятся в верхний регистр

capitalize(): переводит в верхний регистр первую букву только самого первого слова строки

lstrip(): удаляет начальные пробелы из строки

rstrip(): удаляет конечные пробелы из строки

strip(): удаляет начальные и конечные пробелы из строки

ljust(width): если длина строки меньше параметра width, то справа от строки добавляются пробелы, чтобы дополнить значение width, а сама строка выравнивается по левому краю

rjust(width): если длина строки меньше параметра width, то слева от строки добавляются пробелы, чтобы дополнить значение width, а сама строка выравнивается по правому краю

center(width): если длина строки меньше параметра width, то слева и справа от строки равномерно добавляются пробелы, чтобы дополнить значение width, а сама строка выравнивается по центру

find(str[, start [, end]): возвращает индекс подстроки в строке. Если подстрока не найдена, возвращается число -1

replace(old, new[, num]): заменяет в строке одну подстроку на другую

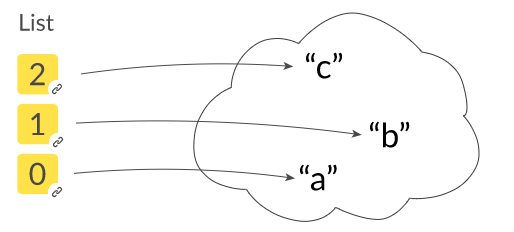
split([delimeter[, num]]): разбивает строку на подстроки в зависимости от разделителя

join(strs): объединяет строки в одну строку, вставляя между ними определенный разделитель.

**Списки**

В python списки хранят упорядоченный набор элементов, которые могут быть разных типов. В примере, указанном выше элементы имеют один и тот же тип int. Не обязательно все элементы должны быть одного типа.

Python размещает элементы списка в памяти, затем размещает указатели на эти элементы. Таким образом, список в Python — это массив указателей**.**



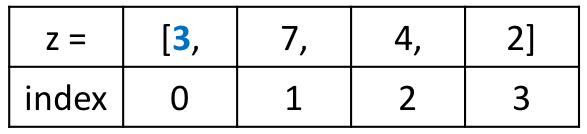
**# Создание списка с разными типам данных**

**heterogenousElements = [3, True, 'Витя', 2.0]**

Этот список содержит int, bool, string и float.

Доступ к элементам списка

Каждый элемент имеет присвоенный ему индекс. Важно отметить, в python индекс первого элемента в списке — 0.



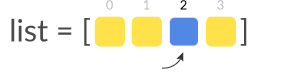
**z = [3, 7, 4, 2] # создаем список**

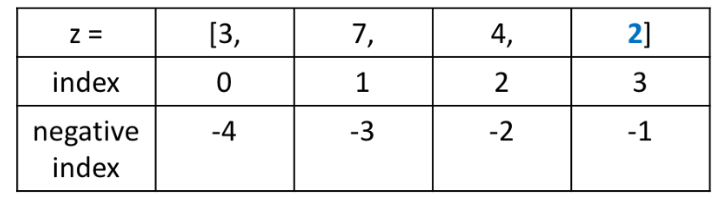
**# обращение к первому элементу списка с индексом 0**

**print(z[0])**

**# элемент с индексом 0 -> 3**

Также поддерживается отрицательная индексация. Отрицательная индексация начинается с конца.





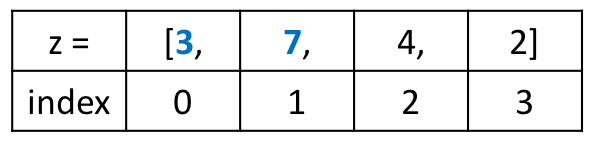
Также можете получить доступ к одному и тому же элементу с использованием положительных индексов. Альтернативный способ доступа к последнему элементу в списке z.

**>>> print(z[3])**

**2**

**Срезы(slice) списка**

Срезы хороши для получения подмножества значений с вашего списка. На примере кода, приведенного ниже, он вернет список с элементами из индекса 0 и не включая индекс 2.



Первый индекс пишется (до : включительно), а последний (после : ) и не учитывается.

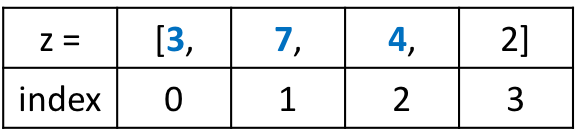
**# Создайте список**

**z = [3, 7, 4, 2]**

**# Вывод элементов с индексом от 0 до 2 (не включая 2)**

**print(z[0:2])**

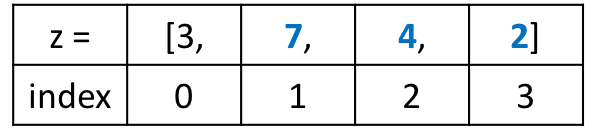
**# вывод: [3, 7]**



**# Все, кроме индекса 3**

**>>> print(z[:3])**

**[3, 7, 4]**



**# начиная с индекса 1 до конца списка**

**>>> print(z[1:])**

**[7, 4, 2]**

Изменение элементов в списке

Списки изменяемы. Это означает, что после создания списка можно обновить его отдельные элементы.

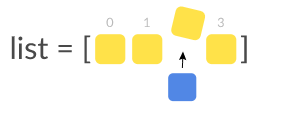
**z = [3, 7, 4, 2] # Создание списка**

**# Изменяем элемент с индексом 1 на строку 'fish'**

**z[1] = 'fish'**

**print(z)**

**[3, 'fish', 4, 2] результат программы**



**Методы и функции списков Рython**

У списков [Python есть разные методы](https://pythonru.com/osnovy/3-python-dlja-data-science-vstroennye-funkcii-i-metody-python), которые помогают в программировании. В этом разделе рассматриваются все методы списков.

Метод index возвращает положение первого индекса, со значением х. В указанном ниже коде, он возвращает назад 0.

**# Создайте список**

**>>> z = [4, 1, 5, 4, 10, 4]**

**>>> print(z.index(4))**

**0**

Метод count работает так, как звучит. Он считает количество раз, когда значение появляется в списке.

**>>> random\_list = [4, 1, 5, 4, 10, 4]**

**>>> print(random\_list.count(4))**

**3**

**Метод sort сортирует и меняет исходный список**.

**z = [3, 7, 4, 2]**

**z.sort()**

**print(z)**

**[2, 3, 4, 7] результат программы**

**# Сортировка и изменение исходного списка от наивысшего к наименьшему**

**z.sort(reverse = True)**

**print(z)**

**[7, 4, 3, 2]**

**# Сортировка списка строками**

**names = ["Стив", "Рейчел", "Майкл", "Адам", "Джессика", "Лестер"]**

**names.sort()**

**print(names)**

**['Адам', 'Джессика', 'Лестер', 'Майкл', 'Рейчел', 'Стив'] результат программы**

**Метод append** добавляет элемент в конец списка. Это происходит на месте.



**z = [7, 4, 3, 2]**

**z.append(3)**

**print(z)**

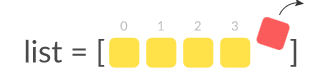
**[7, 4, 3, 2, 3] результат программы**

**Метод remove** удаляет первое вхождение значения в списке.

**z = [7, 4, 3, 2, 3]**

**z.remove(2)**

**print(z)**



Код удаляет первое вхождение значения 2 из списка z.

**[7, 4, 3, 3] результат программы**

**Метод pop** удаляет элемент в указанном индексе. Этот метод также вернет элемент, который был удален из списка. В случае, если вы не указали индекс, он по умолчанию удалит элемент по последнему индексу.

**z = [7, 4, 3, 3]**

**print(z.pop(1))**

**print(z)**

**4**

**[7, 3, 3] - результат программы**

**Метод extend** расширяет список, добавляя элементы. Преимущество над append в том, что вы можете добавлять списки.

Добавим [4, 5] в конец z:

**z = [7, 3, 3]**

**z.extend([4,5])**

**print(z)**

**[7, 3, 3, 4, 5] результат программы**

Эквивалентно коду ниже:

**>>> print([1,2] + [3,4])**

**[7, 3, 3, 4, 5]**

**Метод insert** вставляет элемент перед указанным индексом.

**z = [7, 3, 3, 4, 5]**

**z.insert(4, [1, 2])**

**print(z)**

**[7, 3, 3, 4, [1, 2], 5] результат программы**

**Простые операции над списками**

x in s True если элемент x находится в списке s

x not in s True если элемент x не находится в списке s

s1 + s2 Объединение списков s1 и s2

s \* n , n \* s Копирует список s n раз

len(s) Длина списка s, т.e. количество элементов в s

min(s) Наименьший элемент списка s

max(s) Наибольший элемент списка s

sum(s) Сумма чисел списка s

for i in list() Перебирает элементы слева направо в цикле for

Рассмотрим примеры использования простых операций:

**>>> list1 = [2, 3, 4, 1, 32]**

**>>> 2 in list1 # 2 в list1?**

**True**

**>>> 33 not in list1 # 33 не в list1?**

**True**

**>>> len(list1) # количество элементов списка**

**5**

**>>> max(list1) # самый большой элемент списка**

**32**

**>>> min(list1) # наименьший элемент списка**

**1**

**>>> sum(list1) # сумма чисел в списке**

**42**

**# генератор списков python (list comprehension)**

**>>> x = [i for i in range(10)]**

**>>> print(x)**

**[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]**

**>>> print(list1.reverse()) # разворачивает список**

**[32, 1, 4, 3, 2]**

**Операторы + и \* для списков**

+ объединяет два списка.



**list1 = [11, 33]**

**list2 = [1, 9]**

**list3 = list1 + list2**

**print(list3)**

**[11, 33, 1, 9] результат программы**

\* копирует элементы в списке.

**list4 = [1, 2, 3, 4]**

**list5 = list4 \* 3**

**print(list5)**

**[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4] результат программы**

**Оператор in и not in**

Оператор in проверяет находится ли элемент в списке. При успешном результате он возвращает True, в случае неудачи, возвращает False .



**>>> list1 = [11, 22, 44, 16, 77, 98]**

**>>> 22 in list1**

**True результат программы**

Аналогично not in возвращает противоположный от оператора in результат.

**>>> 22 not in list1**

**False результат программы**

**Итерация по списку с использованием цикла for**

Список — последовательность. Ниже способ, которые вы можете использовать для цикла, чтобы перебрать все элементы списка.

list1 = [1,2,3,4,5]

for i in list1:

print(i, end=" ")

**1 2 3 4 5 результат программы**

**Преобразование списка в строку**

Как преобразовать список в строку?

Для преобразования списка в строку используйте метод join(). В Python это выглядит так:

",".join(["a", "b", "c"]) -> "a,b,c".

Разделитель пишут в кавычках перед join, в список должен состоять из строк.

**mylist = ['spam', 'ham', 'eggs']**

**print(', '.join(mylist))**

**spam, ham, eggs результат программы**

или

**>>> print('\n'.join(mylist))**

**spam**

**ham**

**eggs результат программы**

**list\_of\_ints = [80, 443, 8080, 8081]**

**print(str(list\_of\_ints).strip('[]'))**

**80, 443, 8080, 8081 результат программы**

**>>> print(', '.join(map(str, list\_of\_ints)))**

**80, 443, 8080, 8081**

**>>> print('\n'.join(map(str, list\_of\_ints)))**

**80**

**443**

**8080**

**8081 результат программы**

**Как перевести список в другой формат?**

Иногда требуется перевести список в строку, в словарь или в JSON. Для этого нужно будет вывести список без скобок.

Перевод списка в строку осуществляется с помощью функции join(). На примере это выглядит так:

**>>> fruits = ["яблоко", "груша", "ананас"]**

**>>> print(', '.join(fruits))**

**яблоко, груша, ананас**

В данном случае в качестве разделителя используется запятая.

Словарь в Python — это такая же встроенная структура данных, наряду со списком. Преобразование списка в словарь — задача тоже несложная. Для этого потребуется воспользоваться функцией dict(). Вот пример преобразования:

**>>> elements = [['1', 'a'],['2', 'b'],['3', 'c']]**

**>>> my\_dict = dict(elements)**

**>>> print(my\_dict)**

**{'1': 'a', '2': 'b', '3': 'c'}**

JSON — это JavaScript Object Notation. В Python находится встроенный модуль json для кодирования и декодирования данных JSON. С применением метода json.dumps(x) можно запросто преобразовать список в строку JSON.

**>>> import json**

**>>> json.dumps(['word', 'eye', 'ear'])**

**'["word", "eye", "ear"]'**

**Как узнать индекс элемента в списке?**

Узнать позицию элемента в последовательности списка бывает необходимым, когда элементов много, вручную их не сосчитать, и нужно обращение по индексу. Для того чтобы узнать индекс элемента, используют функцию list.index(x).

**>>> elements = [1, 3, 6, 9, 55]**

**>>> print(elements.index(9))**

**3**

В качестве аргумента передаем значение, а на выходе получаем его индекс.

**Как посчитать количество уникальных элементов в списке?**

Самый простой способ — приведение списка к set (множеству). После этого останутся только уникальные элементы, которые мы посчитаем функцией len():

**>>> words = ["one", "two", "one", "three", "one"]**

**>>> len(set(words))**

**3**

**Как проверить список на пустоту?**

**>>> a = []**

**>>> if not a:**

**print("список пуст!")**

**список пуст!**

**Как создать список числовых элементов с шагом?**

Создание списка числовых элементов с шагом может понадобиться не так и часто, но мы рассмотрим пример построения такого списка.

Шагом называется переход от одного элемента к другому. Если шаг отрицательный, произойдёт реверс массива, то есть отсчёт пойдёт справа налево. Вот так выглядит список с шагом.

**>>> elements = [1, 2, 3, 4, 5, 8, 9, 10, 11, 14, 20]**

**>>> print(elements[0:11:2])**

**[1, 3, 5, 9, 11, 20]**

Еще один вариант — воспользоваться генератором списков:

**>>> elements = [c for c in range(0, 10, 2)] # от 0 (включительно) до 10 с шагом 2**

**>>> print(elements)**

**[0, 2, 4, 6, 8]**

**Как разбить строку на символы?**

Разбиение строки на отдельные символы выполняется несколькими способами:

>>> text = "django"

**# вариант 1**

**>>> list(text)**

**['d', 'j', 'a', 'n', 'g', 'o']**

**# вариант 2**

**>>> [c for c in "text"]**

**['t', 'e', 'x', 't']**

**# вариант 3**

**>>> for c in text:**

**print(c)**

**d**

**j**

**a**

**n**

**g**

**o**

**Как из строки выделить числа?**

Для извлечения чисел из строки можно воспользоваться методами строк:

**>>> str = "h3110 23 cat 444.4 rabbit 11 2 dog"**

**>>> [int(s) for s in str.split() if s.isdigit()]**

**[23, 11, 2]**

Данный пример извлекает только целые положительные числа. Более универсальный вариант – регулярные выражения:

**>>> str = "h3110 23 cat 444.4 rabbit 11 2 dog"**

**>>> import re**

**>>> re.findall(r'\d+', str)**

**['3110', '23', '444', '4', '11', '2']**

**Как перевернуть строку?**

Существует несколько способов перевернуть строку, начиная от классического – запустить цикл в обратной последовательности, выбирая каждый символ с конца и вставляя его в новую строку, и заканчивая срезами – вариант только для Питона.

С помощью среза — самый быстрый способ:

**>>> 'test'[::-1]**

**'tset'**

**Использование reversed() и str.join() :**

**>>> ''.join(reversed('test'))**

**'tset'**

**Как удалить последний символ в строке?**

Как и в предыдущем случае – чтобы убрать последний символ наиболее простым и быстрым вариантом будет использование среза:

**>>> "Some text1"[:-1]**

**'Some text'**

**Как убрать пробелы из строки?**

В случае удаления пробелов со строки может быть два варианта:

1 Обрезать строку так, чтобы удалить с нее первый и последний пробел, такой вариант может пригодиться, когда пользователь случайно поставит пробел в конце введенного текста:

**>>> " Some text ".strip()**

**'Some text'**

2 Удалить со строки все пробелы:

**>>> " Some text".replace(' ', '')**

**'Sometext'**

**Кортежи**

**1 Введение**

**Кортежи** (tuple) в Python – это те же списки за одним исключением. Кортежи неизменяемые структуры данных. Так же как списки они могут состоять из элементов разных типов, перечисленных через запятую. Кортежи заключаются в круглые, а не квадратные скобки.

**>>> a = (10, 2.13, "square", 89, 'C')**

**>>> a**

**(10, 2.13, 'square', 89, 'C')**

Из кортежа можно извлекать элементы и брать срезы:

**>>> a[3]**

**89**

**>>> a[1:3]**

**(2.13, 'square')**

Однако изменять его элементы нельзя:

**>>> a[0] = 11**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**TypeError: 'tuple' object does not support item assignment**

Также у типа tuple нет методов для добавления и удаления элементов.

**2 Особенности работы с кортежами**

Возникает резонный вопрос. Зачем в язык программирования был введен этот тип данных, по-сути представляющий собой неизменяемый список? Дело в том, что иногда надо защитить список от изменений. Преобразовать же кортеж в список, если это потребуется, как и выполнить обратную операцию легко с помощью встроенных в Python функций list() и tuple():

**>>> a = (10, 2.13, "square", 89, 'C')**

**>>> b = [1, 2, 3]**

**>>> c = list(a)**

**>>> d = tuple(b)**

**>>> c**

**[10, 2.13, 'square', 89, 'C']**

**>>> d**

**(1, 2, 3)**

Рассмотрим случай, когда уместно использовать кортежи. В Python изменяемые объекты передаются в функцию по ссылке. Это значит, что не создается копия объекта, а переменной-параметру присваивается ссылка на уже существующий объект. В итоге, если в теле функции объект изменяется, то эти изменения касаются глобального объекта.

**def add\_num(seq, num):**

**for i in range(len(seq)):**

**seq[i] += num**

**return seq**

**origin = [3, 6, 2, 6]**

**changed = add\_num(origin, 3)**

**print(origin)**

**print(changed)**

Данная программа неправильная. Хотя никаких выбросов исключений не произойдет, она содержит логическую ошибку. На выводе получаем:

**[6, 9, 5, 9]**

**[6, 9, 5, 9]**

То есть исходный список был также изменен. Параметр seq содержал ссылку не на свой локальный список, а на список-оригинал. Таким образом, в операторе return здесь нет смыла. Если функция замысливалась как изменяющая глобальный список, то программа должна выглядеть так:

**def add\_num(seq, num):**

**for i in range(len(seq)):**

**seq[i] += num**

**origin = [3, 6, 2, 6]**

**add\_num(origin, 3)**

**print(origin)**

Что делать, если все же требуется не изменять исходный список, а сформировать по нему новый. Задачу можно решить несколькими способами. Во-первых, в функции можно создать локальный список, после чего возвращать его:

**def add\_num(seq, num):**

**new\_seq = []**

**for i in seq:**

**new\_seq.append(i + num)**

**return new\_seq**

**origin = [3, 6, 2, 6]**

**changed = add\_num(origin, 3)**

**print(origin)**

**print(changed)**

Результат:

**[3, 6, 2, 6]**

**[6, 9, 5, 9]**

Исходный список в функции не меняется. Его элементы лишь перебираются.

Второй способ защитить список-оригинал – использовать кортеж. Этот способ более надежный, так как в больших программах трудно отследить, что ни одна функция не содержит команд изменения глобальных данных.

Хотя преобразовывать к кортежу можно как при передаче в функцию, так и в самой функции, лучше сразу делать глобальный список кортежем. Поскольку неизменяемые объекты передаются по значению, а не по ссылке, то в функцию будет поступать копия структуры, а не оригинал. Даже если туда передается оригинал, изменить его невозможно. Можно лишь, как вариант, скопировать его и/или изменить тип, создав тем самым локальную структуру, и делать с ней все, что заблагорассудится.

**def add\_num(seq, num):**

**seq = list(seq)**

**for i in range(len(seq)):**

**seq[i] += num**

**return seq**

**origin = (3, 6, 2, 6)**

**changed = add\_num(origin, 3)**

**print(origin)**

**print(changed)**

Рассмотрим еще примеры кортежей:

**# пустой кортеж**

**empty\_tuple = ()**

**# кортеж из 4-х элементов разных типов**

**four\_el\_tuple = (36.6, 'Normal', None, False)**

**# пример tuple, что содержит вложенные элементы**

**nested\_elem\_tuple = (('one', 'two'), ['three', 'four'], {'five': 'six'}, (('seven', 'eight'), ('nine', 'ten')))**

**print(nested\_elem\_tuple)**

**> (('one', 'two'), ['three', 'four'], {'five': 'six'}, (('seven', 'eight'), ('nine', 'ten')))**

**Списки в кортежах**

Кортежи могут содержать списки, также как списки быть вложенными в другие списки.

**>>> nested = (1, "do", ["param", 10, 20])**

Как вы думаете, можем ли мы изменить список ["param", 10, 20] вложенный в кортеж nested? Список изменяем, кортеж – нет. Если вам кажется, что нельзя, то вам кажется неправильно. На самом деле можно:

**>>> nested[2][1] = 15**

**>>> nested**

**(1, 'do', ['param', 15, 20])**

Примечание. Выражения типа nested[2][1] используются для обращения к вложенным объектам. Первый индекс указывает на позицию вложенного объекта, второй – индекс элемента внутри вложенного объекта. Так в данном случае сам список внутри кортежа имеет индекс 2, а элемент списка 10 – индекс 1 в списке.

Странная ситуация. Кортеж неизменяем, но мы все-таки можем изменить его. На самом деле кортеж остается неизменяемым. Просто в нем содержится не сам список, а ссылка на него. Ее изменить нельзя. Но менять сам список можно.

Чтобы было проще понять, перепишем кортеж так:

**>>> l = ["param", 10, 20]**

**>>> t = (1, "do", l)**

**>>> t**

**(1, 'do', ['param', 10, 20])**

Кортеж содержит переменную-ссылку. Поменять ее на другую ссылку нельзя. Но кортеж не содержит самого списка. Поэтому его можно менять как угодно:

**>>> l.pop(0)**

**'param'**

**>>> t**

**(1, 'do', [10, 20])**

Однако такой номер не проходит с неизменяемыми типами:

**>>> a = "Kat"**

**>>> t = (a, l)**

**>>> t**

**('Kat', [10, 20])**

**>>> a = "Bat"**

**>>> t**

**('Kat', [10, 20])**

Они передаются в кортеж, как и в функцию – по значению. То есть их значение копируется в момент передачи.

Вывод, определим некоторые возможности кортежей:

* они упорядочены по позициям;
* tuple могут хранить и содержать внутри себя объекты любых типов (и даже составных);
* доступ к элементам происходит по смещению, а не по ключу;
* в рамках этой структуры данных определены все операции, основанные на применении смещения (индексирование, срез);
* кортежи поддерживают неограниченное количество уровней вложенности;
* кортежи хранят указатели на другие объекты, а значит их можно представлять, как массивы ссылок;
* они позволяют очень просто менять местами значения двух переменных.

Характеристики кортежей:

* Неизменяемость — именно это свойство кортежей, порой, может выгодно отличать их от списков.
* Скорость — кортежи быстрее работают. По причине неизменяемости кортежи хранятся в памяти особым образом, поэтому операции с их элементами выполняются заведомо быстрее, чем с компонентами списка.
* Безопасность — неизменяемость также позволяет им быть идеальными кандидатами на роль констант. Константы, заданные кортежами, позволяют сделать код более читаемым и безопасным.
* Использование tuple в других структурах данных — кортежи применимы в отдельных структурах данных, от которых python требует неизменяемых значений. Например, ключи словарей (dicts) должны состоять исключительно из данных immutable-типа.