# Лекция 2 "Управляющие конструкции и списки"

## часть 2 Списки и кортежи


### Финансовый университет при Правительстве РФ, лектор С.В. Макрушин


v 0.8 27.07.2021

## Разделы:



-


- к оглавлению


In [1]:

```
# загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v1.css")
HTML(html.read().decode('utf-8'))
```

Out[1]:

# Списки


- к оглавлению


### Введение

**Списки и кортежи** (list, tuple) - это нумерованные наборы объектов.

- Каждый элемент набора содержит лишь **ссылку на объект**.

- По этой причине списки и кортежи могут содержать **объекты произвольного типа данных** и иметь неограниченную степень вложенности.
- Позиция элемента в наборе задается **целочисленным индексом**. Нумерация элементов, как и в строках, **начинается с 0**, а не с 1.

In [3]:

```
lst = [3, 7, 5]  # создаем список
```

In [4]:

```
lst[0] # получаем элемент по индексу (так же, как и у строк)
```

Out[4]:

```
3
```

In [5]:

```
lst[2]
```

Out[5]:

```
5
```

In [6]:

```
lst
```

Out[6]:

```
[3, 7, 5]
```

In [8]:

```
lst[0] = 13 # Изменяем элемент по индексу
# для неизменяемых аналогов списка, в т.ч. для строк, эта операция недоступна
```

In [9]:

```
print(lst)
```

```
[13, 7, 5]
```

In [10]:

```
lst
```

Out[10]:

```
[13, 7, 5]
```

## Создание списка

- к оглавлению

In [11]:

```
type(lst)
```

Out[11]:

```
list
```

In [12]:

```
# создание пустого списка с помощью конструктора
lst_1 = list()
lst_1
```

Out[12]:

```
[]
```

In [13]:

```
# список с заданными значениями
lst_2 = [2, True, 'my string', '7'] # значение в списке могут быьт разных типов
lst_2
```

Out[13]:

```
[2, True, 'my string', '7']
```

In [15]:

```
lst_3 = [] # создание пустого списка

lst_3.append(1) # добавление элемента в конец списка
lst_3.append(False)
lst_3.append('my_stirng')
lst_3
```

Out[15]:

```
[1, False, 'my_stirng']
```

In [16]:

```
# list() может создать список из любого итерируемого объекта:
lst_1 = list('Hello world')
lst_1
```

Out[16]:

```
['H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
```

In [17]:

```
# длинный и некрасивый вариант
# создания списка из итерируемого объекта (в данном случае - строки)
lst_1v2 = []
for e in 'Hello world':
    lst_1v2.append(e)
lst_1v2
```

Out[17]:

```
['H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
```

In [2]:

```
# встроенная функция range() генерирует последовательность чисел
lst_5 = list()
for i  in range(5, 10):
    lst_5.append(i)
    print(i, end=' ')
print('\n', lst_5)
```

```
5 6 7 8 9
 [5, 6, 7, 8, 9]
```

In [20]:

```
# список может создаваться на основе функций или объектов, которые возвращают последователь
lst_5 = list(range(5, 10))
lst_5
```

Out[20]:

```
[5, 6, 7, 8, 9]
```

К чему могут приводить множественные ссылки на изменяемые объект:

In [22]:

```
#ld = [8] # изменяемый объект
ld2 = [8]
lst_t1 = [ld2, ld2, ld2, ld2]
lst_t1
```

Out[22]:

```
[[8], [8], [8], [8]]
```

In [30]:

```
ld2.append(9)
lst_t1
```

Out[30]:

```
[[8, 9], [8, 9], [8, 9], [8, 9]]
```

>

# Вложенные списки

- к оглавлению

In [23]:

```
# создание вложенных списков:
lst_6 = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
lst_6
```

Out[23]:

```
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```

Обращение к элементам вложенных списков:

In [24]:

```
lst_6[1]
```

Out[24]:

```
[0, 1, 0]
```

In [25]:

```
lst_6[1][0]
```

Out[25]:

```
0
```

Изменение вложенных списков:

In [26]:

```
lst_6[1][2] = 3
lst_6
```

Out[26]:

```
[[1, 0, 0], [0, 1, 3], [0, 0, 1]]
```

**Вложенные списки** - это полноценные независимые объекты класса `list` , поэтому:

- вложенные списки могут менять размер независимо от других списков
  - это может приводить к превращению вложенных списков к "не прямоугольному" виду
- объекты, хранящиеся во вложенных списках, могут иметь произвольный тип.

In [28]:

```
lst_6[2].append('My string')
lst_6
```

Out[28]:

```
[[1, 0, 0], [0, 1, 3], [0, 0, 1, 'My string', 'My string']]
```

In [29]:

```
# длина и тип элементов вложенных списков не обязаны быть одинаковыми:
lst_7 = [[1, 0, 0], True, [0, 1, 0, 5], ['Ivanov', 'Ivan']]
lst_7
```

Out[29]:

```
[[1, 0, 0], True, [0, 1, 0, 5], ['Ivanov', 'Ivan']]
```

In [30]:

```
len(lst_7) # длина основного списка
```

Out[30]:

```
4
```

In [31]:

```
len(lst_7[0]) # длина вложенного списка с индексом 0
```

Out[31]:

```
3
```

In [32]:

```
len(lst_7[2]) # длина вложенного списка с индексом 2
```

Out[32]:

```
4
```

# >

---

## Копирование списков


- к оглавлению


При создании списка в переменной **сохраняется ссылка на объект, а не сам объект**. Это обязательно следует учитывать при групповом присвоаивании. Групповое присваиванне безопасно использовать для чисел и строк, но для списков это может приводить к ошибкам.

In [33]:

```
lst_8 = lst_9 = [1, 2, 3] # lst_8 и lst_9 ссылаются на один и тот же список!
print('lst_8:', lst_8)
print('lst_9:', lst_9)
```

```
lst_8: [1, 2, 3]
lst_9: [1, 2, 3]
```

In [34]:

```
lst_8[0] = 13 # изменение списка на который ссылаются и lst_8, и lst_9!
print('lst_8:', lst_8)

print('lst_9:', lst_9)
```

```
lst_8: [13, 2, 3]
lst_9: [13, 2, 3]
```

In [35]:

```
lst_8 is lst_9 # проверка на совпадение ссылок на списки
```

Out[35]:

```
True
```

Создание поверхностной копии списка.

In [36]:

```
lst_from = [1, 2, 3, 4, 5] # создание списка
lst_from
```

Out[36]:

```
[1, 2, 3, 4, 5]
```

In [37]:

```
lst_to = list(lst_from) # создание нового списка на основе объекта, который возвращает посл
lst_to
```

Out[37]:

```
[1, 2, 3, 4, 5]
```

In [38]:

```
lst_from is lst_to # проверка на совпадение ссылок на списки
```

Out[38]:

```
False
```

In [39]:

```
lst_from == lst_to # проверка на равенство значений списков
```

Out[39]:

```
True
```

In [41]:

```
# изменение исходного списка не влияет на его копию:
lst_from[0] = 13
print('lst_from:', lst_from)
print('lst_to:', lst_to)
```

```
lst_from: [13, 2, 3, 4, 5]
lst_to: [1, 2, 3, 4, 5]
```

Поверхностное копирование вложенных списков не приводит к копированию вложенных списков:

In [42]:

```
lst_from2 = [1, 2, [3, 4, 5]] # создание списка
lst_to2 = list(lst_from2)
lst_to2
```

Out[42]:

```
[1, 2, [3, 4, 5]]
```

In [43]:

```
lst_from2 is lst_to2 # проверка на совпадение ссылок на списки
```

Out[43]:

```
False
```

In [44]:

```
lst_from2[0] = 13
print(lst_from2)
print(lst_to2)
```

```
[13, 2, [3, 4, 5]]
[1, 2, [3, 4, 5]]
```

In [46]:

```
lst_from2[2] is lst_to2[2] # вложенные списки являются одним объектом, на которой имеется д
```

Out[46]:

```
True
```

In [47]:

```
lst_from2[2][0] = 13 # изменение списка на который ссылаются и lst_from2[2], и lst_to2[2]!
print('lst_from2:', lst_from2)
print('lst_to2:', lst_to2)
```

```
lst_from2: [13, 2, [13, 4, 5]]
lst_to2: [1, 2, [13, 4, 5]]
```

**Создание глубокой копии списков**

In [48]:

```
import copy # подключаем модуль copy

lst_to2_deep  =  copy.deepcopy(lst_from2) #  делаем полную копию списка
lst_to2
```

Out[48]:

```
[1, 2, [13, 4, 5]]
```

In [49]:

```
lst_from2[2] is lst_to2_deep[2] # вложенные списки НЕ совпадают
```

Out[49]:

```
False
```

In [50]:

```
lst_from2[2][1] = 44
print(lst_from2)
print(lst_to2_deep)
```

```
[13, 2, [13, 44, 5]]
[13, 2, [13, 4, 5]]
```

# >

---

# Операции над списками: индексация и срезы


- к оглавлению

In [51]:

```
lst = [10, 2, 13, 4, 55]
# определение длины списка:
len(lst) # len() - встроенная функция (ее не нужно импортировать, она доступна всегда)
```

Out[51]:

```
5
```

In [52]:

```
lst[len(lst)] # ошибка! Обращение к не существующему элементу списка.
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-52-2975f9683a62> in <module>
----> 1 lst[len(lst)] # ошибка! Обращение к не существующему элементу списка.

IndexError: list index out of range
```

In [53]:

```
lst[len(lst)-1] # обращение к последнему элементу списка
```

Out[53]:

```
55
```

Как и в строках, допустимо использование отрицательных индексов:

In [54]:

```
lst[-1] # при помощи отрицательных индексов можно обращаться к элементам списка "с конца"
```

Out[54]:

```
55
```

In [55]:

```
lst[-5]
```

Out[55]:

```
10
```

In [56]:

```
lst[-6] # ошибка! Допустимы отрицательные индексы не превышающие по модулю длину списка.
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-56-0c16925aa5f3> in <module>
----> 1 lst[-6] # ошибка! Допустимы отрицательные индексы не превышающие по м
одулю длину списка.

IndexError: list index out of range
```

**Извлечение среза**

Списки поддерживают **операцию извлечения среза**, которая возвращает указанный фрагмент списка. Формат операции: `[<Начало>:<Конец>:<Шаг>]`

Все параметры в операции среза являются необязательными. В частности, если:

- НЕ указан параметр `<Начало>`, то используется значение 0
- НЕ указан параметр `<Конец>`, то возвращается фрагмент до конца списка. Следует также заметить, что элемент с индексом, указанном в этом параметре, не входит в возвращаемый фрагмент.
- НЕ указан параметр `<Шаг>`, то используется значение 1. При этом второе двоеточие в срезе не используется.

В качестве значения параметров можно указать отрицательные значения.

In [57]:

```
lst
```

Out[57]:

```
[10, 2, 13, 4, 55]
```

Срезы работают также как и для строк:

In [58]:

```
lst[1:] # список без первого элемента
```

Out[58]:

```
[2, 13, 4, 55]
```

In [59]:

```
lst[:2] # элемент с индексом 2 не входит в диапазон
```

Out[59]:

```
[10, 2]
```

In [67]:

```
lst[1:4] # возвращаются элементы с индексами 1, 2 и 3
```

Out[67]:

```
[2, 13, 4]
```

In [60]:

```
lst[:-1] # список без последнего элемента
```

Out[60]:

```
[10, 2, 13, 4]
```

In [69]:

```
lst[:] # срез с пустыми параметрами создает поверхностную копию списка
```

Out[69]:

```
[10, 2, 13, 4, 55]
```

In [61]:

```
lst[::-1] # шаг -1 позволяет получить список в обратном порядке
```

Out[61]:

```
[55, 4, 13, 2, 10]
```

## >

---

# Операции над списками: изменение списка


- к оглавлению


`append(<Объект>)` - добавляет один объект в конец списка. Метод изменяет текущий список и ничего не возвращает.

Изменение размера списков Python за счет добавления элементов в конец (или удаления последних элементов списка) является наиболее эффективной (с точки зрения затраты вычислительных ресурсов) операцией изменения списков.

In [62]:

```
lst4 = [3, 7, 1]
```

In [63]:

```
lst4.append(8)
lst4
```

Out[63]:

```
[3, 7, 1, 8]
```

In [64]:

```
# append добавляет объект как единичный элемент списка
# даже если аргумент является списком (или другим итерируемым объектом)
lst4.append([1, 2, 3])
lst4
```

Out[64]:

```
[3, 7, 1, 8, [1, 2, 3]]
```

`extend(<Последовательность>)` - добавляет элементы последовательности в конец списка. Метод изменяет текущий список и ничего не возвращает.

In [65]:

```
lst5 = [6, 11, 7, 6, 8]
lst5.extend([1, 2, 3])
lst5
```

Out[65]:

```
[6, 11, 7, 6, 8, 1, 2, 3]
```

In [66]:

```
lst4 = [3, 7, 1]
lst5 = [6, 11, 7, 8]
```

In [67]:

```
# конкатенация (соединение) 2х списков:
lst4 + lst5 # создается новый конкатенированный список
```

Out[67]:

```
[3, 7, 1, 6, 11, 7, 8]
```

In [68]:

```
# как создать список представляющий многократное повторение другого списка?
# очевидный вариант:
lst4 + lst4 + lst4 + lst4 + lst4
```

Out[68]:

```
[3, 7, 1, 3, 7, 1, 3, 7, 1, 3, 7, 1, 3, 7, 1]
```

In [69]:

```
# использование специального синтаксиса
# (по определению умножение a * b это сложение a самого с собой b раз):
lst4 * 5
```

Out[69]:

```
[3, 7, 1, 3, 7, 1, 3, 7, 1, 3, 7, 1, 3, 7, 1]
```

In [70]:

```
# красивый способ создания списка состоящего из одинаковых элементов:
lst_mul = [7] * 7
lst_mul
```

Out[70]:

```
[7, 7, 7, 7, 7, 7, 7]
```

In [71]:

```
lst6 = [0, 7]
lst6 += lst4 # список присоединяется к текущему
lst6
```

Out[71]:

```
[0, 7, 3, 7, 1]
```

`insert (<Индекс>, <Объект>)` - добавляет один объект в указанную позицию. Остальные элементы смещаются. Метод изменяет текущий список и ничего не возвращает. Метод `insert()` позволяет добавить только один объект. Чтобы добавить несколько объектов, можно воспользоваться операцией присваивания значения срезу.

In [84]:

```
lst4 = [3, 7, 1]
```

In [72]:

```
lst4.insert(2, 13)
lst4
```

Out[72]:

```
[3, 7, 13, 1]
```

In [73]:

```
lst4[2]
```

Out[73]:

```
13
```

С помощью среза можно изменить фрагмент списка. Если срезу присвоить пустой список, то элементы, попавшие в срез, будут удалены.

In [75]:

```
lst4
```

Out[75]:

```
[3, 7, 13, 1]
```

In [74]:

```
lst4[1:1]
```

Out[74]:

```
[]
```

In [76]:

```
# вставка нескольких элементов при помощи среза (без потери существующих элементов):
lst4[1:1] = ['a', 'b', 'c']
lst4
```

Out[76]:

```
[3, 'a', 'b', 'c', 7, 13, 1]
```

In [77]:

```
# изменение значений с помощью среза:
lst4[1:3] = ['A', 'B']
lst4
```

Out[77]:

```
[3, 'A', 'B', 'c', 7, 13, 1]
```

In [78]:

```
# вставка нескольких элементов при помощи среза (с замещением двух существующих элементов):
lst4[1:3] = ['X', 'Y', 'Z']
lst4
```

Out[78]:

```
[3, 'X', 'Y', 'Z', 'c', 7, 13, 1]
```

In [79]:

```
# удаление элементов при помощи среза:
lst4[4:6] = []
lst4
```

Out[79]:

```
[3, 'X', 'Y', 'Z', 13, 1]
```

`pop([<Индекс>])` - удаляет элемент, расположенный по указанному индексу, и возвращает его.

- Если индекс не указан, то удаляет и возвращает последний элемент списка.
- Если элемента с указанным индексом нет или список пустой, возбуждается исключение IndexError.

In [80]:

```
lst4
```

Out[80]:

```
[3, 'X', 'Y', 'Z', 13, 1]
```

In [81]:

```
lst4.pop() # без параметров извлекается последний элемент из списка
# (эквивалентно использованию параметра -1)
```

Out[81]:

```
1
```

In [82]:

```
lst4
```

Out[82]:

```
[3, 'X', 'Y', 'Z', 13]
```

In [83]:

```
cur_el = lst4.pop(1)
print(cur_el)
```

```
X
```

In [84]:

```
lst4
```

Out[84]:

```
[3, 'Y', 'Z', 13]
```

**Инструкция `del`** может удалять из списка как единичные элементы, так и элементы, получаемые при помощи среза

In [85]:

```
del lst4[2]
lst4
```

Out[85]:

```
[3, 'Y', 13]
```

In [86]:

```
del lst4[:2]
lst4
```

Out[86]:

```
[13]
```

In [87]:

```
# удаление переменной lst4
# (объект списка будет удален только после того, как будет удалена последняя ссылка на него
del lst4
```

In [88]:

```
lst4
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-88-d3c082ee31c3> in <module>
----> 1 lst4

NameError: name 'lst4' is not defined
```

`remove (<Значение>)` - удаляет первый элемент, содержащий **указанное значение**. Если элемент не найден, возбуждается исключение ValueError. Метод изменяет текущий список и ничего не возвращает.

In [89]:

```
lst5
```

Out[89]:

```
[6, 11, 7, 8]
```

In [90]:

```
lst5.remove(7) # удаляет первое встретившееся значение 7
```

In [92]:

```
lst5
```

Out[92]:

```
[6, 11, 8]
```

In [93]:

```
lst5.remove(77)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-93-9be72156b20d> in <module>
----> 1 lst5.remove(77)

ValueError: list.remove(x): x not in list
```

In [94]:

```
# повторение списка:
lst5 * 3
```

Out[94]:

```
[6, 11, 8, 6, 11, 8, 6, 11, 8]
```

Проверка на вхождение элемента в список

Оператор `in` осуществляет проверку на вхождение элемента в список. Если элемент входит в список, то возвращается значение `True`, в противном случае - `False`. Оператор `in` не дает никакой информации о местонахождении элемента внутри списка.

In [95]:

```
lst6 = [6, 11, 7, 6, 8, 1, 2, 3]
```

In [96]:

```
# проверка на вхождение элемента в список:
3 in lst6
```

Out[96]:

```
True
```

In [97]:

```
111 in lst6
```

Out[97]:

```
False
```

>

---

# Операции над списками: поиск, сортировка и обход

- к оглавлению

**Поиск элемента в списке**

Метод `index()` позволяет узнать индекс элемента с определенным значением.

Формат метода: `index(<Значение> [,<Начало> [,<Конец>]])` . Метод `index()` возвращает индекс элемента, имеющего указанное значение.

- Если значение не входит в список, то возбуждается исключение ValueError.
- Если второй и третий параметры не указаны, то поиск будет производиться с начала списка.

In [98]:

```
lst7 = [1, 2, 1, 2, 1]
```

In [99]:

```
len(lst7)
```

Out[99]:

```
5
```

In [100]:

```
lst7.index(1)
```

Out[100]:

```
0
```

In [114]:

```
lst7.index(2)
```

Out[114]:

```
1
```

In [101]:

```
lst7.index(2, 2)
```

Out[101]:

```
3
```

In [102]:

```
lst7.index(2, 2, 4)
```

Out[102]:

```
3
```

In [103]:

```
lst7.index(2, 2, 3)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-103-73faa58fe6ae> in <module>
----> 1 lst7.index(2, 2, 3)

ValueError: 2 is not in list
```

In [105]:

```
# Подсчет количества элементов в списке:
lst7.count(1)
```

Out[105]:

```
3
```

**Обход списка**

In [106]:

```
for el in lst6:
    print(el, end=' ')
```

```
6 11 7 6 8 1 2 3
```

In [107]:

```
# функция enumerate позволяет удобно обходить список и оперировать с текущим индексом элеме
for i, el in enumerate(lst6):
    print(f'индекс: {i}, значение: {el}')
```

```
индекс: 0, значение: 6
индекс: 1, значение: 11
индекс: 2, значение: 7
индекс: 3, значение: 6
индекс: 4, значение: 8
индекс: 5, значение: 1
индекс: 6, значение: 2
индекс: 7, значение: 3
```

**Агрегирующие функции**

In [109]:

```
lst5
```

Out[109]:

```
[6, 11, 8]
```

In [110]:

```
sum(lst5)
```

Out[110]:

```
25
```

In [111]:

```
min(lst5)
```

Out[111]:

```
6
```

In [112]:

```
max(lst5)
```

Out[112]:

```
11
```

Функции работают с любыми итерируемыми объектами, для которых допустимы необходимые операции ( + и < соответственно):

In [113]:

```
min('Hello world')
```

Out[113]:

```
' '
```

In [114]:

```
max('Hello world')
```

Out[114]:

```
'w'
```

**Сортировка**

Отсортировать список позволяет метод `sort()` . Метод имеет формат `sort([key=None] [, reverse=fa1se])` . Все параметры являются необязательными. Метод изменяет текущий список и ничего не возвращает.

- использование аргумента `key` будет рассмотрено в других лекциях

In [115]:

```
lst8 = [2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
```

In [116]:

```
lst8.sort() # изменяет текущий список
lst8
```

Out[116]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [117]:

```
lst8.sort(reverse=True)
lst8
```

Out[117]:

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Метод `sort()` сортирует сам список и не возвращает никакого значения. В некоторых случаях необходимо получить отсортированный список, а текущий список оставить без изменений. Дпя этого следует воспользоваться функцией `sorted()` .

Функция имеет формат: `sorted(<Последовательность>[, key=None] [, reverse=False])`

- В первом параметре указывается список, который необходимо отсортировать.
- Остальные параметры эквивалентны параметрам метода sort().

In [118]:

```
lst8 = [2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
```

In [119]:

```
sorted(lst8)
```

Out[119]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [120]:

```
lst8 # исходный список не поменялся
```

Out[120]:

```
[2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
```

**Инвертирование списка**

In [121]:

```
lst8
```

Out[121]:

```
[2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
```

In [122]:

```
lst8[::-1]
```

Out[122]:

```
[5, 1, 3, 9, 8, 6, 4, 10, 7, 2]
```

In [123]:

```
lst8
```

Out[123]:

```
[2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
```

In [124]:

```
lst8.reverse() # операция переворачивает список изменяя исходный объект
lst8
```

Out[124]:

```
[5, 1, 3, 9, 8, 6, 4, 10, 7, 2]
```

Если необходимо изменить порядок следования и получить новый список, то следует воспользоваться функцией `reversed(<Последовательность>)`. Функция возвращает итератор, который можно преобразовать в список с помощью функции `list()`.

In [125]:

```
rit8 = reversed(lst8)
rit8
```

Out[125]:

```
<list_reverseiterator at 0x1c7239e1b08>
```

In [126]:

```
lst8
```

Out[126]:

```
[5, 1, 3, 9, 8, 6, 4, 10, 7, 2]
```

In [127]:

```
for e in lst8[::-1]:
    print(e)
```

```
2
7
10
4
6
8
9
3
1
5
```

In [128]:

```
for e in reversed(lst8):
    print(e)
```

```
2
7
10
4
6
8
9
3
1
5
```

In [129]:

```
for e in reversed(lst8):
    print(e)
```

```
2
7
10
4
6
8
9
3
1
5
```

In [130]:

```
lst8_rev = list(rit8)
lst8_rev
```

Out[130]:

```
[2, 7, 10, 4, 6, 8, 9, 3, 1, 5]
```

>

# Кортежи

- к оглавлению

Кортежи - не изменяемые списки.

In [131]:

```
tu1 = (1, True, 'My string')
tu1
```

Out[131]:

```
(1, True, 'My string')
```

In [132]:

```
type(tu1)
```

Out[132]:

```
tuple
```

In [133]:

```
tu1[0] = 11 # Ошибка!
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-133-dfc8d57b11b8> in <module>
----> 1 tu1[0] = 11 # Ошибка!

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

In [134]:

```
tu1[0]
```

Out[134]:

```
1
```

In [135]:

```
for e in tu1:
    print(e)
```

```
1
True
My string
```

In [136]:

```
lst9 = [1, True, 'My string']
tu2 = tuple(lst9)
tu3 = tuple(lst9[2])

print(tu2)
print(tu3)
```

```
(1, True, 'My string')
('M', 'y', ' ', 's', 't', 'r', 'i', 'n', 'g')
```

In [150]:

```
tuple(range(4, 10, 2))
```

Out[150]:

```
(4, 6, 8)
```

In [137]:

```
tu4 = (5,)
tu4
```

Out[137]:

```
(5,)
```

In [138]:

```
len(tu3)
```

Out[138]:

```
9
```

In [139]:

```
list(reversed(tu3))
```

Out[139]:

```
['g', 'n', 'i', 'r', 't', 's', ' ', 'y', 'M']
```

In [140]:

```
len(tu4)
```

Out[140]:

```
1
```

**>**

In [ ]:

```

```

# Задание к Лекции 3

По книге Н. Прохоренок:

Глава 8 Списки и кортежи, Глава 9 Словари и множества

По книге М. Саммерфильд:

Глава 3 Типы коллекций