

# Лекция 5

Язык программирования Python.

Хайрулин Сергей Сергеевич

email: [s.khairulin@g.nsu.ru](mailto:s.khairulin@g.nsu.ru), [s.khayrulin@gmail.com](mailto:s.khayrulin@gmail.com)

Ссылка на [материалы](#)

# План

- Лекции/практические занятия
  - Тест
- Дифференцированный зачет в конце семестра
  - Защита задания

# Литература

## Начальный уровень

- Mark Pilgrim. Dive into Python - <http://www.diveintopython.net/>
- Марк Лутц. Изучаем Python, 4-е издание // Символ-Плюс 2011.
- ...

## Стандарт/Документация

- PEP-8 - <https://www.python.org/dev/peps/pep-0008/>
- <https://www.python.org/>
- <https://github.com/python/cpython>

## Экспертный уровень

- Лучано Рамальо: Python. К вершинам мастерства
- Mitchell L. Model. Bioinformatics Programming Using Python // O'Reilly 2010.

# Версии Python

- Python 2 вышел 2010 году последняя версия 2.7.16 - исправлялись только баги(ошибки) с января 2020 года поддержка прекращена.
- Python 3 появился в 2008, является актуальной версией языка. Текущая стабильная версия 3.8.5 -> в предрелиз 3.9, в разработке 3.10
  - Python 3 не гарантирует совместимости кода с Python 2

# План занятия

- Коллекция
- Индексация
- tuple
- list
  - Срезы
  - Списковые включения
- Операции над списками
- Практика

# Инициализация аргументов функции

В случае инициализации аргументов функции мутабельными объектами, важно учитывать что. Инициализация будет произведена один раз а не каждый раз при вызове функции

# Инициализация аргументов функции

```
In [1]: def foo(l=[]):  
...:     l.append(1)  
...:
```

```
In [2]: def foo(l=[]):  
...:     l.append(1)  
...:     print(l)  
...:
```

```
In [3]: foo()  
[1]
```

```
In [4]: foo()  
[1, 1]
```

```
In [5]: foo()  
[1, 1, 1]
```

```
In [6]: foo([1,2,3])  
[1, 2, 3, 1]
```



# Массив

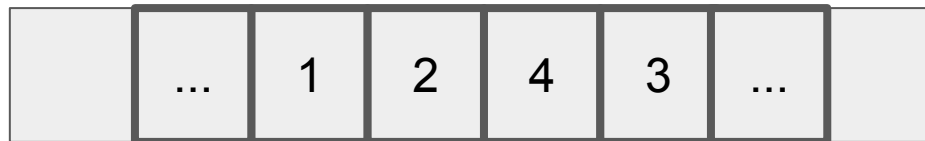
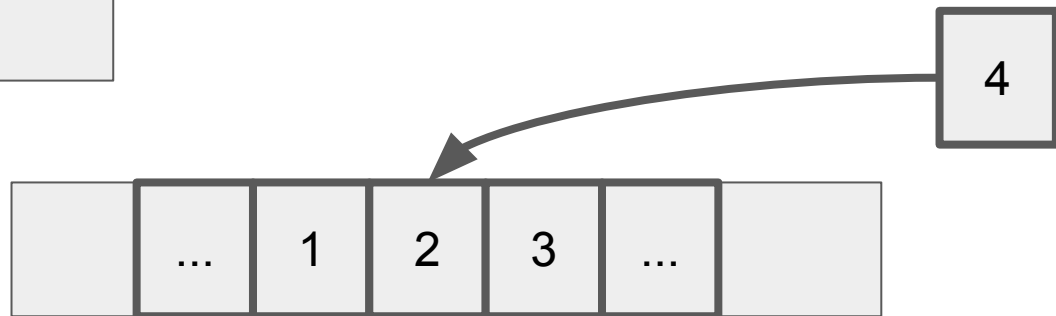
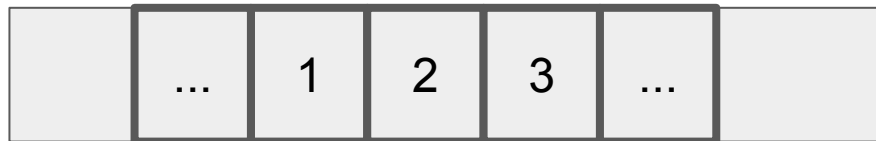
Фундаментальная структура данных, позволяющая хранить некоторый объем непересекающихся данных в непрерывном куске оперативной памяти. Обычно массивы представляют к каждому отдельной сущности массива через порядковый индекс - это операция называется **индексация**. Поддерживается большинством языков программирования. Python не исключение.

# Динамический массив

Обычно при создании массива предполагается, что массив будет иметь фиксированную длину, которая не меняется. Соответственно стандартный массив не поддерживает операции вставки и удаления элемента. Поэтому были введены специализированный тип данных как **динамический массив**.

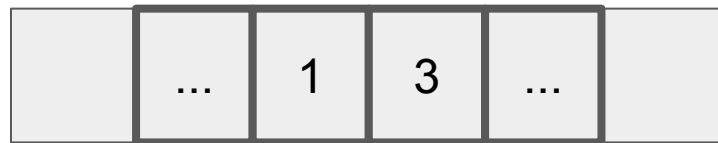
# Массив

RAM



# Массив

RAM



# Коллекции

- **Коллекция** — программный объект, содержащий в себе, тем или иным образом, набор значений одного или различных типов, и позволяющий обращаться к этим значениям.
- **Коллекция** позволяет записывать в себя значения и извлекать их.
- **Назначение коллекции** — служить хранилищем объектов и обеспечивать доступ к ним.

# Индексация

Некоторые коллекции поддерживают операцию индексацию

В python операция обозначается [...] - аргументом операции является индекс, который указывает какой элемент коллекции нужно взять

Синтаксис:

```
collection_name[index]
```

где collection\_name - переменная содержащая ссылку на коллекцию

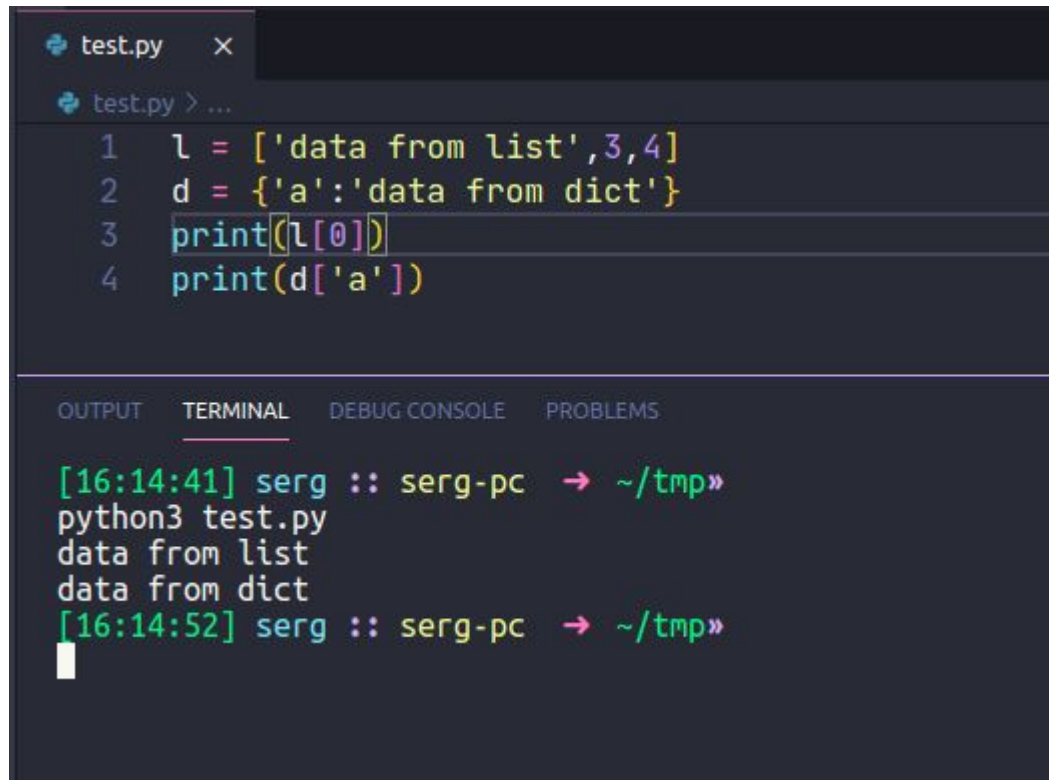
index - указатель на элемент коллекции

# Индексация

Для коллекций, сохраняющих порядок элементов, таких как **list**, **tuple**, **range** - индексом является порядковый номер элемента в коллекции. **ИНДЕКСАЦИЯ В PYTHON ДЛЯ ТАКИХ КОЛЛЕКЦИЙ НАЧИНАЕТСЯ С 0!** Также в python поддерживаются отрицательные индексы (см. примеры).

Для **dict**, **set**, ... - индексом является значение ключа элемента (об этом на др лекции).

# Индексация



```
test.py x
test.py > ...
1 l = ['data from list',3,4]
2 d = {'a':'data from dict'}
3 print(l[0])
4 print(d['a'])
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[16:14:41] serg :: serg-pc → ~/tmp»
python3 test.py
data from list
data from dict
[16:14:52] serg :: serg-pc → ~/tmp»
```



# Python - collection package

## `collections` — Container datatypes

Source code: [Lib/collections/\\_\\_init\\_\\_.py](#)

This module implements specialized container datatypes providing alternatives to Python's general purpose built-in containers, `dict`, `list`, `set`, and `tuple`.

|                           |                                                                      |
|---------------------------|----------------------------------------------------------------------|
| <code>namedtuple()</code> | factory function for creating tuple subclasses with named fields     |
| <code>deque</code>        | list-like container with fast appends and pops on either end         |
| <code>ChainMap</code>     | dict-like class for creating a single view of multiple mappings      |
| <code>Counter</code>      | dict subclass for counting hashable objects                          |
| <code>OrderedDict</code>  | dict subclass that remembers the order entries were added            |
| <code>defaultdict</code>  | dict subclass that calls a factory function to supply missing values |
| <code>UserDict</code>     | wrapper around dictionary objects for easier dict subclassing        |
| <code>UserList</code>     | wrapper around list objects for easier list subclassing              |
| <code>UserString</code>   | wrapper around string objects for easier string subclassing          |

*Deprecated since version 3.3, will be removed in version 3.10:* Moved [Collections Abstract Base Classes](#) to the `collections.abc` module. For backwards compatibility, they continue to be visible in this module through Python 3.9.

<https://docs.python.org/3.8/library/collections.html>

# Python built-in collection

## Sequence Types — list, tuple, range

There are three basic sequence types: lists, tuples, and range objects. Additional sequence types tailored for processing of `binary data` and `text strings` are described in dedicated sections.

<https://docs.python.org/3.8/library/stdtypes.html#sequence-types-list-tuple-range>

# tuple

```
test.py x
test.py > ...
1 l1 = 1,2,3
2 l2 = (1,2,3)
3 print(l1)
4 print(l2)
5
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[15:10:10] serg :: serg-pc → ~/tmp»
cd /home/serg/tmp ; /usr/bin/env /usr/bin/python3
g/tmp/test.py
(1, 2, 3)
(1, 2, 3)
[15:10:14] serg :: serg-pc → ~/tmp»
```

```
test.py x
test.py > ...
1 l1 = 1,2,3
2 l1[0] = 123
```

**Exception has occurred: TypeError**  
'tuple' object does not support item assignment  
File "/home/serg/tmp/test.py", line 2, in <module>  
l1[0] = 123

<https://docs.python.org/3.8/library/stdtypes.html#tuple>

# list

```
test.py x
test.py > ...
1 l = [1,2,3,4]
2 print(l)

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

[15:12:37] serg :: serg-pc → ~/tmp»
cd /home/serg/tmp ; /usr/bin/env /usr/bin/python3
g/tmp/test.py
[1, 2, 3, 4]
[15:12:43] serg :: serg-pc → ~/tmp»
```

```
test.py x
test.py > ...
1 l = [1,2,3,4]
2 print(l)
3 l[2] = 3.14
4 print(l)

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

[15:15:11] serg :: serg-pc → ~/tmp»
cd /home/serg/tmp ; /usr/bin/env /usr/bin/python3
[1, 2, 3, 4]
[1, 2, 3.14, 4]
[15:15:16] serg :: serg-pc → ~/tmp»
```

# range - объект

Позволяет создавать генератор последовательности с заданными стартовым конечным (не включается) и шагом значениями. При этом последовательность не генерируется моментально.

**range**(begin, end, step)

```
In [16]: [1,2,3,4]
Out[16]: [1, 2, 3, 4]
```

```
In [17]: range(10)
Out[17]: range(0, 10)
```

```
In [18]: range(1, 10)
Out[18]: range(1, 10)
```

```
In [19]: range(1, 10, 2)
Out[19]: range(1, 10, 2)
```

```
In [20]: list(range(10))
Out[20]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [21]: list(range(1, 10))
Out[21]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [22]: list(range(1, 10, 2))
Out[22]: [1, 3, 5, 7, 9]
```

## range - объект

```
In [23]: range(-100)
Out[23]: range(0, -100)
```

```
In [27]: list(range(10, 1, -1))
Out[27]: [10, 9, 8, 7, 6, 5, 4, 3, 2]
```

# list

| Operation                                       | Result                                                                                                                        |
|-------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <code>s[i] = x</code>                           | item <code>i</code> of <code>s</code> is replaced by <code>x</code>                                                           |
| <code>s[i:j] = t</code>                         | slice of <code>s</code> from <code>i</code> to <code>j</code> is replaced by the contents of the iterable <code>t</code>      |
| <code>del s[i:j]</code>                         | same as <code>s[i:j] = []</code>                                                                                              |
| <code>s[i:j:k] = t</code>                       | the elements of <code>s[i:j:k]</code> are replaced by those of <code>t</code> (1)                                             |
| <code>del s[i:j:k]</code>                       | removes the elements of <code>s[i:j:k]</code> from the list                                                                   |
| <code>s.append(x)</code>                        | appends <code>x</code> to the end of the sequence (same as <code>s[len(s):len(s)] = [x]</code> )                              |
| <code>s.clear()</code>                          | removes all items from <code>s</code> (same as <code>del s[:]</code> ) (5)                                                    |
| <code>s.copy()</code>                           | creates a shallow copy of <code>s</code> (same as <code>s[:]</code> ) (5)                                                     |
| <code>s.extend(t)</code> or <code>s += t</code> | extends <code>s</code> with the contents of <code>t</code> (for the most part the same as <code>s[len(s):len(s)] = t</code> ) |
| <code>s *= n</code>                             | updates <code>s</code> with its contents repeated <code>n</code> times (6)                                                    |
| <code>s.insert(i, x)</code>                     | inserts <code>x</code> into <code>s</code> at the index given by <code>i</code> (same as <code>s[i:i] = [x]</code> )          |
| <code>s.pop([i])</code>                         | retrieves the item at <code>i</code> and also removes it from <code>s</code> (2)                                              |
| <code>s.remove(x)</code>                        | remove the first item from <code>s</code> where <code>s[i] == x</code> (3)                                                    |
| <code>s.reverse()</code>                        | reverses the items of <code>s</code> in place (4)                                                                             |

# len

Встроенная функция [len](#) позволяет выяснить текущую длину коллекции и тип коллекции не имеет значения.

```
In [12]: l = [1,2,3]

In [13]: m = {'key1':'value1', 'key2':'value2'}

In [14]: len(l)
Out[14]: 3

In [15]: len(m)
Out[15]: 2
```



# append

Добавляет новый элемент в конец списка. Довольно эффективная процедура, так как при этом список **почти всегда** не реаллоцирует память.

# insert

В случае если необходимо вставить элемент в середину или в начало списка, можно воспользоваться этой процедурой, **но** в этом случае список памяти будет реалоцированна, что может привести к потере производительности. Реалокация - процесс перераспределения памяти.

```
list.insert(i, x)
```

Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

# index

Поиск элемента по значению в последовательности - вернет индекс элемента в списке. Если элемента нет то будет порождено исключение

```
list.index(x[, start[, end]])
```

Return zero-based index in the list of the first item whose value is equal to *x*. Raises a `ValueError` if there is no such item.

The optional arguments *start* and *end* are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.

# remove

Удаление элемента из списка по значению. Также в случае отсутствия такого элемента в списке будет порождено исключение

```
list.remove(x)
```

Remove the first item from the list whose value is equal to x. It raises a `ValueError` if there is no such item.

# pop

Извлечь элемент из списка и вернуть его. В случае если мы извлекаем не последний элемент процесс также приведет к релокации памяти.

```
list.pop(i)
```

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

# Списки арифметика (конкатенация)

Списки поддерживают операцию сложения

```
In [1]: l = [1,2,3,4]

In [2]: l1 = [5,6,7,8]

In [3]: l2 = l + l1

In [4]: l2
Out[4]: [1, 2, 3, 4, 5, 6, 7, 8]

In [5]: l += l1

In [6]: l
Out[6]: [1, 2, 3, 4, 5, 6, 7, 8]

In [7]:
```

# Индексация списков

Для списков применима индексация с отрицательными значениями индекса при этом значению -1 будет соответствовать последний элемент списка, -2 предпоследний то есть

```
In [7]: l = [1,2,3]

In [8]: l[-1]
Out[8]: 3

In [9]: l[-1] == l[len(l) - 1]
Out[9]: True
```

# Индексация списков

При обращении по индексу превосходящему длину списка будет сгенерировано исключение

```
In [10]: l[100]
```

```
-----  
IndexError                                Traceback (most recent call last)  
<ipython-input-10-e2a0c2623844> in <module>  
----> 1 l[100]
```

```
IndexError: list index out of range
```



# Создание списков

Список можно создать несколькими способами

1. Явно
2. Списковые включения
3. Ключевое слово `list`

# Явно

Через использование конструкции

[element1, element2, ..., elementN]

```
In [16]: [1,2,3,4]  
Out[16]: [1, 2, 3, 4]
```

# Списковые включения

Более короткий синтаксис, который позволяет создать новый список на основе значений существующего списка.

```
[item for item in collection if condition ]
```

Условный оператор может отсутствовать.

# Списковые включения

```
test.py x
test.py > ...

1  # How to create list
2  #1. loops
3  l = [] # empty list
4  for i in range(10):
5      l.append(i)
6
7  print(l)
8
9  # list comprehension
10 l = [i for i in range(10)]
11 print(l)
```

# Списковые включения

```
test.py x
test.py > ...

1  # How to create list
2  #1. loops
3  l = [] # empty list
4  for i in range(10):
5      l.append(i)
6
7  print(l)
8
9  # list comprehension
10 l = [i for i in range(10)]
11 print(l)
```

# Списковые включения

```
1 # inner list comprehension
2 l = [[j for j in range(i)] for i in range(10)]
3 for row in l:
4     print(row)
```

OUTPUT

TERMINAL

DEBUG CONSOLE

PROBLEMS

```
[17:07:56] serg :: serg-pc → ~/tmp»
python3 test.py
[]
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4, 5]
[0, 1, 2, 3, 4, 5, 6]
[0, 1, 2, 3, 4, 5, 6, 7]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

# Списковые включения

```
test.py > ...  
1  # inner list comprehension  
2  l = [i for i in range(10) if i % 2 == 0]  
3  print(l)
```

---

OUTPUT   TERMINAL   DEBUG CONSOLE   PROBLEMS

```
[17:09:16] serg :: serg-pc → ~/tmp»  
python3 test.py  
[0, 2, 4, 6, 8]  
[17:09:18] serg :: serg-pc → ~/tmp»  
█
```

# Ключевое слово list

Позволяет создавать списки в том числе десериализованные из другого типа последовательностей

```
In [34]: list('abcd')
Out[34]: ['a', 'b', 'c', 'd']

In [35]: list({'1':'2'})
Out[35]: ['1']

In [36]: list((1,2,3))
Out[36]: [1, 2, 3]
```



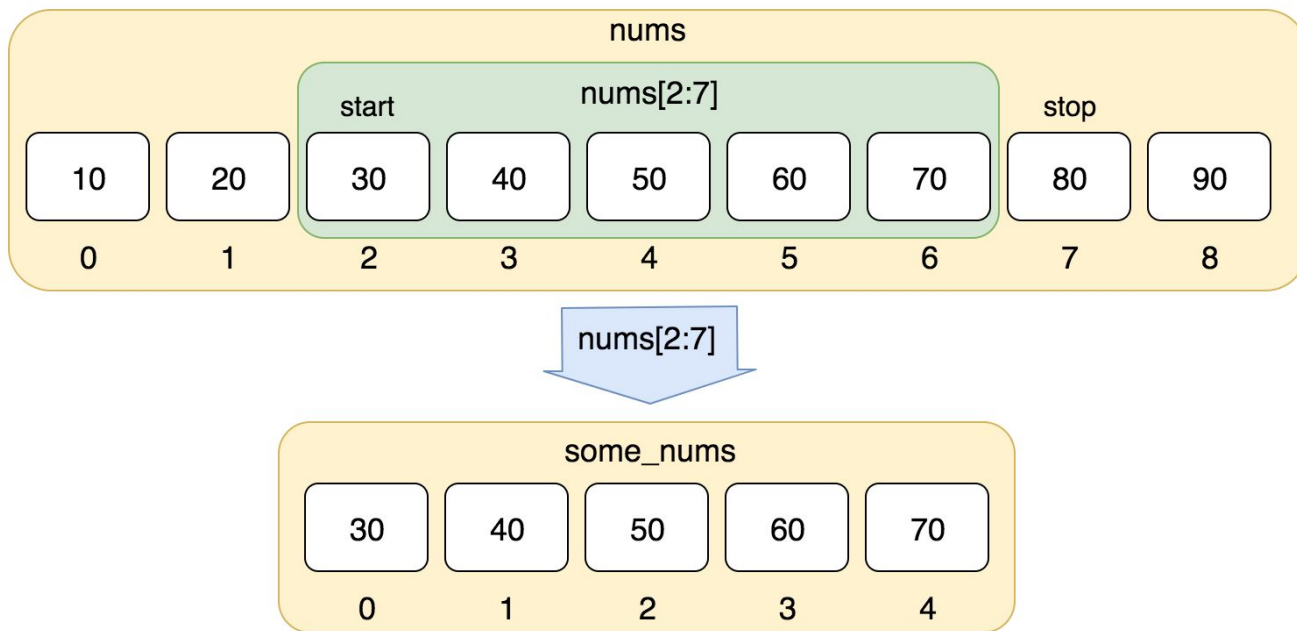
# Срезы (slice)

Расширенная операция индексации, над сохраняющими порядок коллекциями, которая позволяет делать выборки из них. Параметры задаются значениями индексов правым (**который включается в выборку**) и левым (**не включается в выборку**). Синтаксис

`collection_name[start:stop:step]` -> вернет список в котором первым элементов будет элемент `collection_name[start]`

Значение индексов могут быть  $< 0$ . Значение `step` - шаг может отсутствовать по умолчанию он равен 1.

# Срезы (slice)



# Срезы (slice)

```
1 l = [i for i in range(10)]
2
3 print(l)
4 print(l[1:7])
5 print(l[4:])
6 print(l[:5])
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[16:43:49] serg :: serg-pc → ~/tmp»
python3 test.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6]
[4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4]
[16:43:50] serg :: serg-pc → ~/tmp»
```

# Tricks

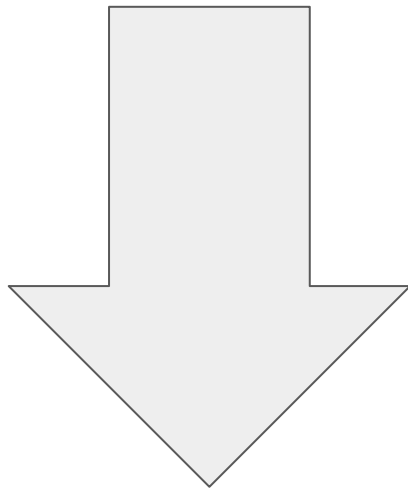
Самый простой способ перевернуть список

```
In [7]: l  
Out[7]: [1, 2, 3, 4, 5, 6, 7, 8]  
  
In [8]: l[::-1]  
Out[8]: [8, 7, 6, 5, 4, 3, 2, 1]
```

Взять каждый второй элемент

```
In [9]: l[::2]  
Out[9]: [1, 3, 5, 7]
```

# Практика



1. Создайте список числовых значений от 0 до 100 (через циклы и генераторы)
2. Создайте список квадратов элементов предыдущего списка (через циклы и генераторы)
3. Создайте список, состоящий из четных элементов предыдущего списка (через циклы и генераторы)
4. Вот строка

'rewlkdfsklgjdflkjglkdsfjgkldfsjglkjeroitewuiotujdigjsdfklg;kl sdfgkl;jsdfkl;gjldk;sfjgjlk;sdfjlk;gjsdfl;kgl;kdsfjgl;kjsdfl;kgj;l;sdfkjg;lkjsdflbvjdfslkgklkrewjhtiowerjutioerutopiytuilyhjsdfl;kghjl;sdkf;gjdfffffffffkgjlkdfjglkasjdfoitweigheripjgiernglisjdfkjlghsdfkj;l;hgkljasdhfglk;hsdfkjlg hlk;sdfhg;kljsdflkgjlk;sdfjgl;ksdfjl;kgjsdfl;kjglk;sdfjgkjsdfl;kgjs;dlkfjgoiw3eujtio34wuytiergoijherjhlgjsdflkjgkl;dfjgkl;sdfjkl;gjsdf;lkg;l;lsjeriotuerl;kjdsfkl;jgh;lksdfjg;lksdfjg;lksdfkjg;lkjreopyulidsjfl;kghjs;ldkjg;lkjlr5l;h;kljyhkl;rirtiriiiiiiiiiiiierwtsj;kldfjg;lksdfjgl;ksdjfl;gj;lksdfjg;lk' -

- удалите из нее все повторяющиеся буквы и выведете строку уникальных букв
5. Какая буквенная подпоследовательность одинаковых символов самая длинная
  6. Напишите функцию которая будет удалять заданную букву из строки и протестируйте ее на вышеприведенной строке

7. Вот список чисел -  
2,3,3,45,4,23,43,54,34,5,32,423,4,23542354,3422,243,4,3,3,254,5643,3233,3,3,4,  
43,2,423,3,3,45,5,43,2,1,4,34234,34,3,342,23,4543,534,32423,23,4,4,4,3,423,324  
5,23,3,34254,235,234,5,235,4,345,235,23,5523,5,234,52,67,756,76,57,345,23,31,  
7,8,56,346,345,756,4343,754,674,8,568,9,65,34,3,5474,5687,56,2,3 - вычислите  
сумму этой последовательности
8. Найдите наибольший/наименьший элемент предыдущего списка
9. Отсортируйте предыдущий список
10. Напишите программу, которая спрашивает пользователя как много чисел  
Фибоначчи нужно сгенерировать а затем генерирует их
11. сгенерируйте матрицу как список списков (через циклы и генераторы)
12. Напишите функцию транспонирования матрицы
13. Напишите функцию сложения матриц
14. Напишите функцию умножения матриц

# Лабораторная работы #2

1. Напишите функцию решения системы линейных уравнений методом Гаусса. Коэффициента уравнения задаются матрицей вектор неизвестных - вектором соответственно. [Метод Гаусса](#)