

The logo consists of a red square with the text 'NEWPROLAB' in white, stacked vertically. The 'O' in 'PRO' is replaced by a circle with a diagonal slash. The logo is surrounded by several overlapping horizontal bars in green, blue, and orange. The background is a dark blue grid with various geometric shapes: a large orange circle in the top right, a blue circle in the top right, a small orange circle in the center, a blue circle in the bottom left, a green square in the bottom right, a red triangle in the bottom right, a blue square in the middle right, and several orange and blue lines of varying lengths and orientations.

NEW  
PRO  
LAB

# PYTHON: ОСНОВЫ

Николай Марков  
@enchantner

NEWPROLAB.COM

# ЧТО ЕЩЕ ЗА PYTHON?

- Язык общего назначения
- Интерпретируемый
- С динамической (“утиной”) типизацией
- Автоматическое управление памятью и Garbage Collector
- Эталонная реализация - CPython (также существуют Pyру, Jython, IronPython)
- Все - объект

# ИНТЕРПРЕТАТОР (REPL)

*~\$ python3*

Python 3.6.4 (default, Mar 9 2018, 23:15:12)

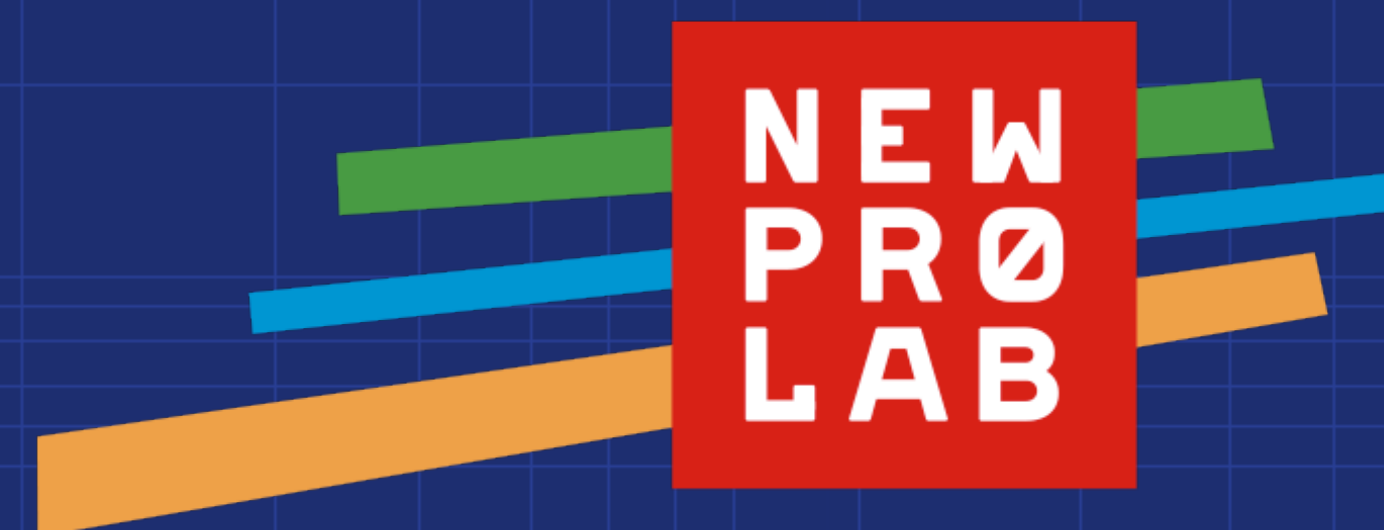
[GCC 4.2.1 Compatible Apple LLVM 9.0.0 (clang-900.0.39.2)] on  
darwin

Type "help", "copyright", "credits" or "license" for more information.

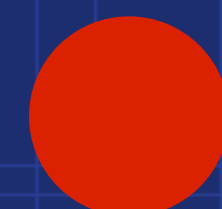
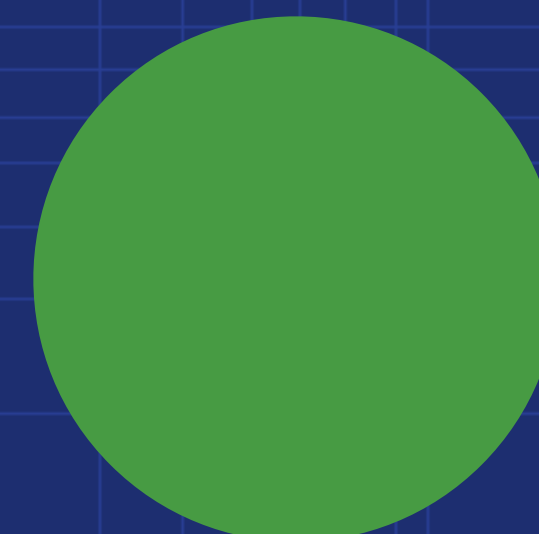
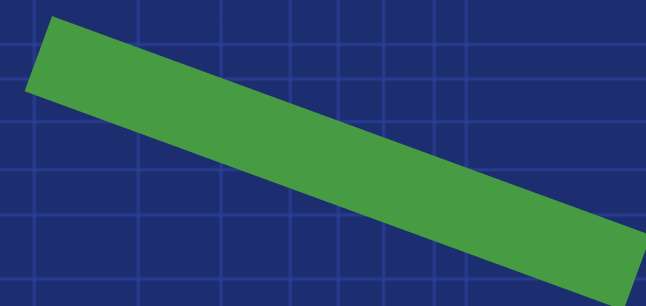
*>>>*

*~\$ pip install ipython*

*~\$ ipython*



# Данные и функции






# ОСНОВНЫЕ ТИПЫ ДАННЫХ

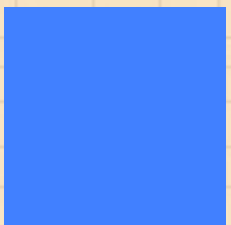


- bool (логический, True/False)
- string (unicode)/bytes (строковый)
- int/float/complex (числовой)

## ОСНОВНЫЕ ТИПЫ КОНТЕЙНЕРОВ

- list (список)
  - tuple (кортеж, неизменяемый список)
  - set (множество)
  - dict (словарь)
- 

<https://docs.python.org/3/library/datatypes.html>



# ОБЪЯВЛЕНИЯ ДАННЫХ

`s = 'abc'` # строка

`s = "abc"` # тоже строка `n = 2` # integer

`n = 3.5` # float

`l = []` # пустой список `l=[1,2,3]` # тоже список

`l = ['a', 2, None]` # список объектов разных типов

`t = (2,)` # кортеж из одного элемента (запятая!)

`d = {}` # пустой словарь

`d = {'a': 1, 'b': 2}` # тоже словарь

`s = set([1, 2])` # множество, еще работает `{1, 2}`



# ОБЪЯВЛЕНИЯ ФУНКЦИЙ

```
def add_numbers(a, b):  
    # здесь тело функции  
    return a + b
```

```
def apply_to_three(f):  
    # принимает функцию как аргумент  
    return f(3)
```


---

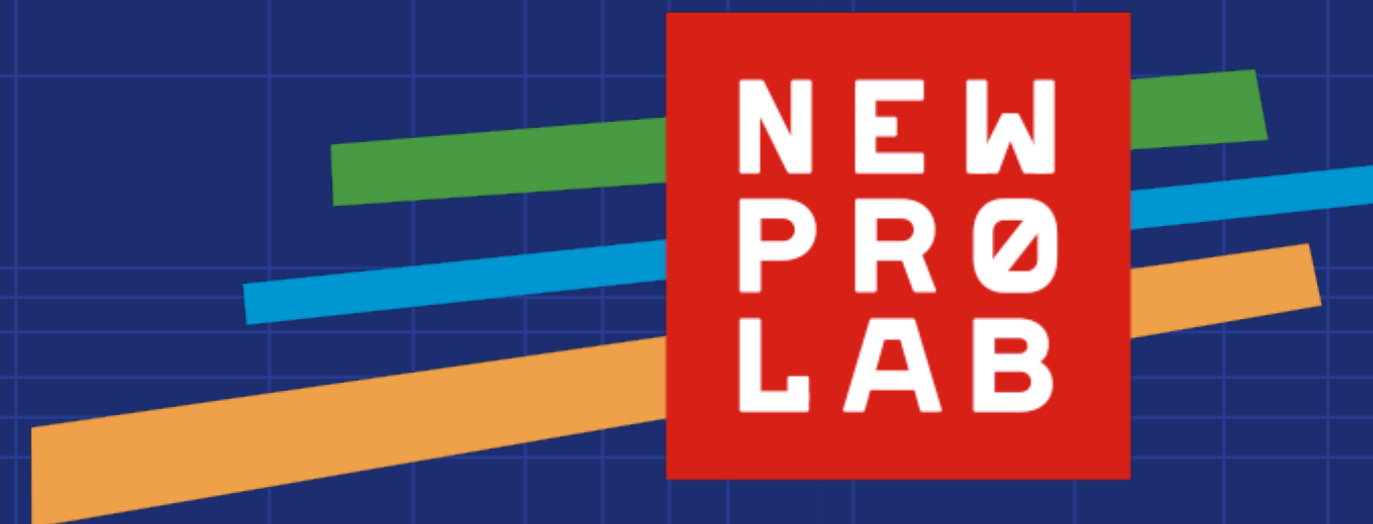
```
def main():  
    # здесь тело функции  
    return 0
```

```
if __name__ == "__main__":  
    main()
```

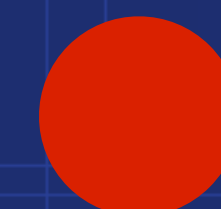
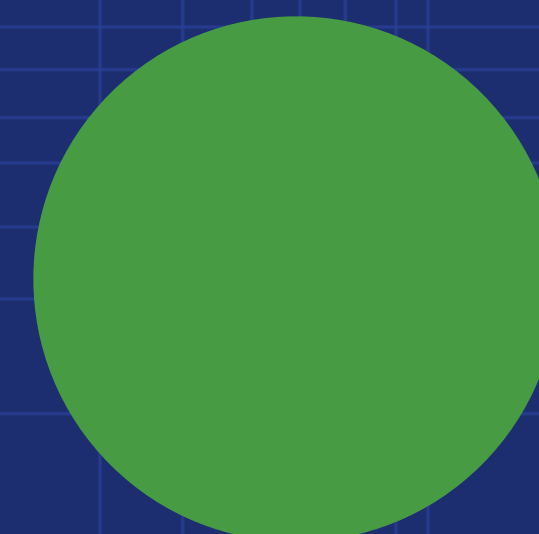
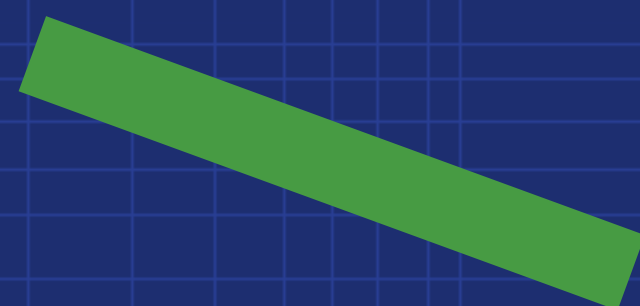
```
apply_to_three(lambda a: a + 5)
```

безымянная функция





# Немного линейной алгебры





# СПИСОК

# Список - непрерывный кусок памяти!

# <https://wiki.python.org/moin/TimeComplexity>

**len(l)** # посчитать длину списка или строки

**sum(l)** # просуммировать элементы списка

**sorted(l)** # вернуть отсортированный список

**sorted(l, reverse=True)** # в обратном порядке

**max(l)** # максимальный элемент; **min(l)** # минимальный

**l.append(x)** # добавить элемент в конец

**l.extend(L)** # расширить один список другим

**l.insert(i, x)** # вставить элемент в позицию i

**l.remove(x)** # удалить первый элемент со значением x

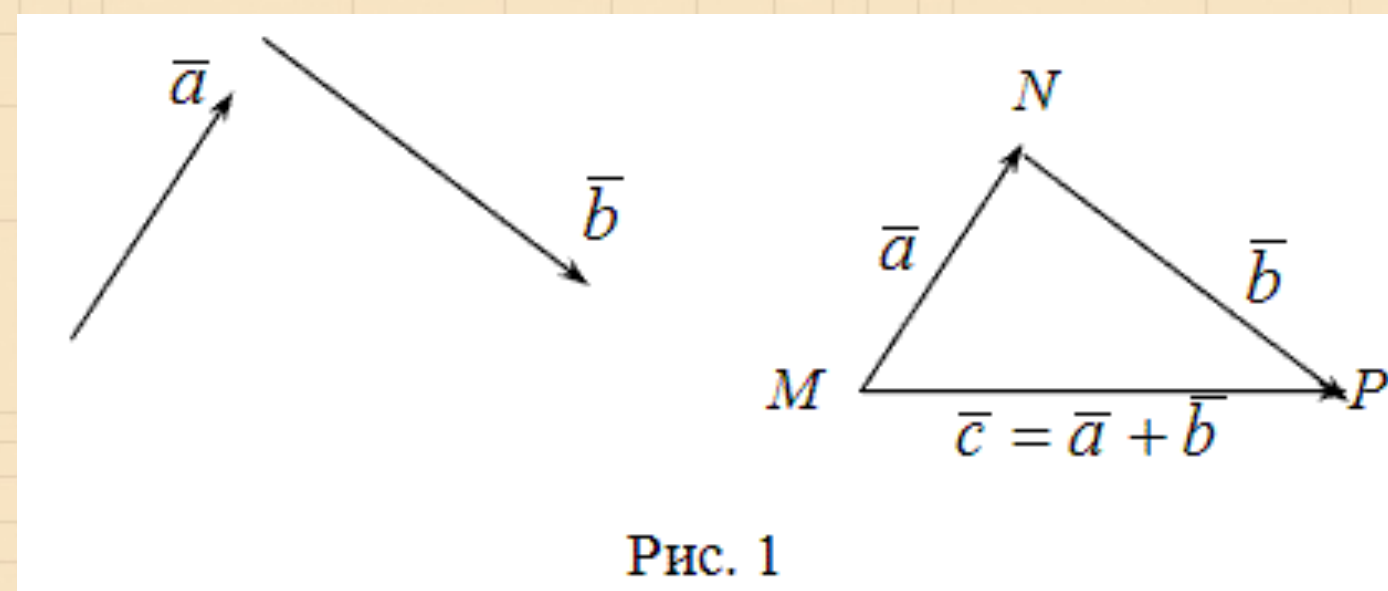
**l.count(x)** # посчитать число элементов со значением x

# ВЕКТОРЫ - НАШЕ ВСЕ

$a = [1, 2]$   
 $b = [2, 1]$

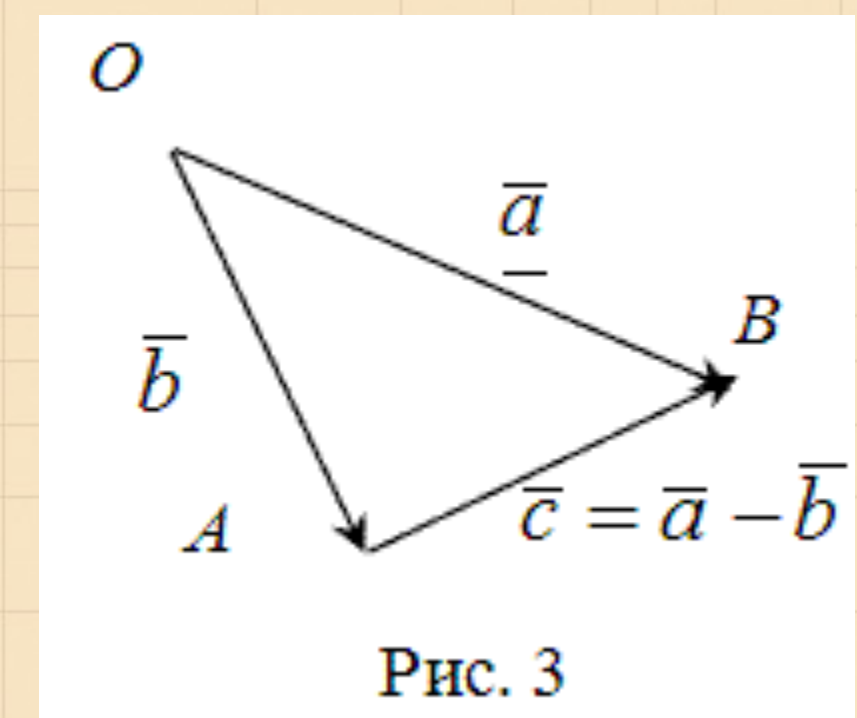
# Как сложить?

```
def vector_add(v1, v2):  
    ...  
    return res
```



# Как вычесть?

```
def vector_sub(v1, v2):  
    ...  
    return res
```



for, zip(), sum()



# ВЕКТОРЫ - НАШЕ ВСЕ

# Как умножить на скаляр?

```
def scalar_mul(v, s):  
    ...  
    return res
```

# Как найти длину вектора? (корень из суммы квадратов)

```
def magnitude(v):  
    ...  
    return res
```

# Скалярное произведение векторов  
(сумма покомпонентных произведений)

```
def dot(v1, v2):  
    ...  
    return res
```

import math



# ВЕКТОРЫ - НАШЕ ВСЕ

# Евклидово расстояние?

(корень из суммы квадратов разностей)

```
def distance(v1, v2):
```

```
    ...
```

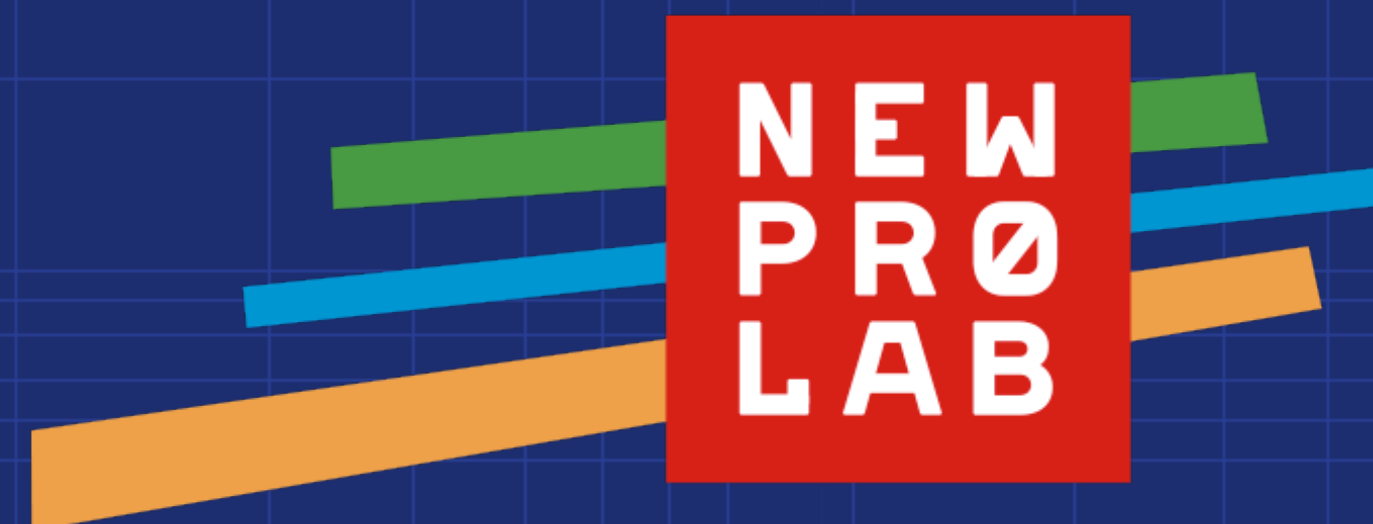
```
    return res
```

$$P = \sqrt{\sum_{i=1}^N (A_i - B_i)^2}$$

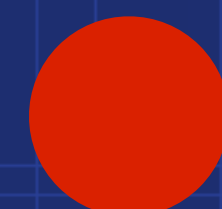
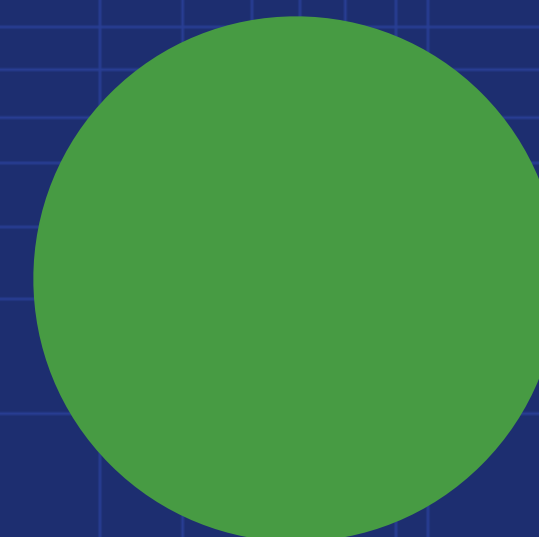
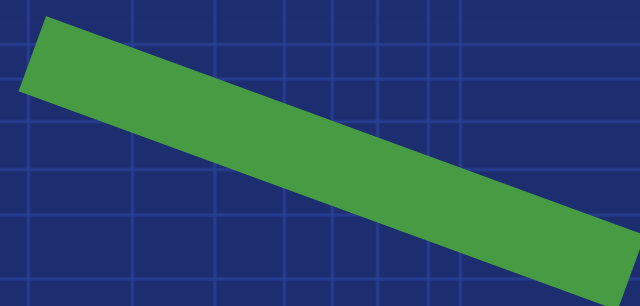
# НЕБОЛЬШОЕ НАПОМИНАНИЕ

Большая часть кода, который мы сегодня пишем,  
непригодна для продакшена.

А теперь продолжим.



# Немного статистики





# БАЗОВЫЕ СРЕДНИЕ

```
def mean(v):  
    ...  
    return res
```

(среднее  
арифметическое)

```
def median(v):  
    ...  
    return res
```

(медиана,  
серединное  
значение)

```
def mode(v):  
    ...  
    return res
```

(мода, самое  
часто  
встречающееся  
значение)

if, %, collections.Counter

# ДИСПЕРСИЯ И СТАНДАРТНОЕ ОТКЛОНЕНИЕ

$$D_x = \frac{\sum_{i=1}^N (x_i - M_x)^2}{N - 1}$$



# МНОЖЕСТВА (SET'Ы)

- # Вставка и проверка наличия элемента - гораздо
- # быстрее, чем в списке!
- # Все элементы уникальны
- # Элементы сортируются в Python 3.5+, но на это полагаться не стоит
- # Поддерживают математические операции
- s.add(x)** # добавить элемент ко множеству
- s.remove(x)** # удалить элемент из множества
- s.pop(x)** # удалить элемент и вернуть его
- x in s** # проверка наличия элемента
- x not in s** # проверка отсутствия элемента



# МНОЖЕСТВА (SET'Ы)

**`s1.issubset(s2)`** # является ли множество подмножеством?

**`s1.issuperset(s2)`** # надмножеством?

**`s1.union(s2)`** # объединение множеств

**`s1.intersection(s2)`** # пересечение множеств

**`s1.difference(s2)`** # разность множеств

## СОКРАЩЕНИЯ

**`s1<=s2`** # можно также `s1<s2`

**`s1>=s2`** # можно также `s1>s2`

**`s1 | s2`**

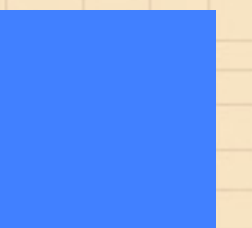
**`s1 & s2`**

**`s1 - s2`**

# ЕСТЬ ДВА СПИСКА...

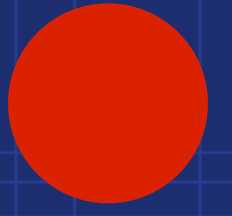
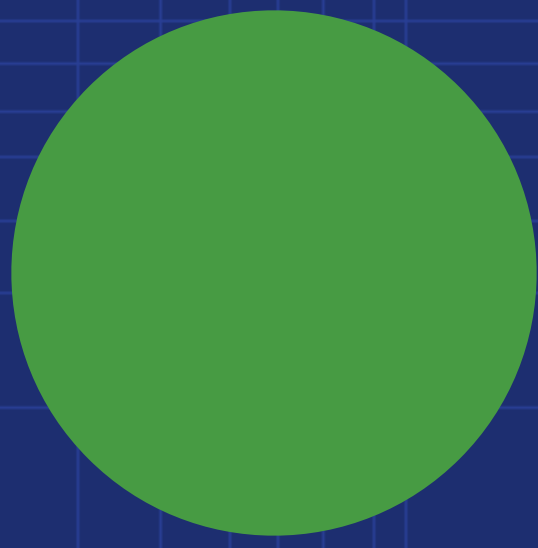
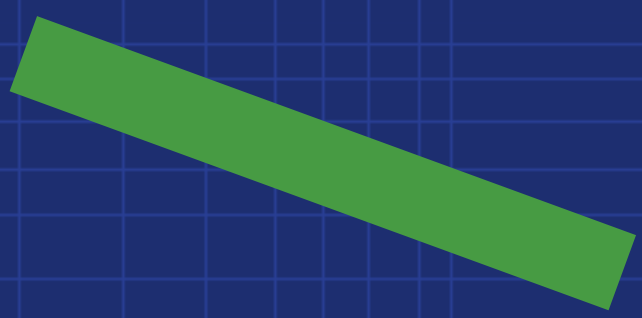


Как посчитать, что добавилось, а что удалилось?





# Внешние источники данных





# РАБОТА СО СТРОКАМИ

# s - строка или список

s[0] # нулевой элемент (индексация с нуля)

s[2:4] # элементы 2 и 3

s[1:8:2] # элементы 1, 3, 5, 7 (шаг 2)

s[-1] # обратный индекс - последний элемент



## РАЗДЕЛЕНИЕ/ОБЪЕДИНЕНИЕ

s.split("a") # разделяем строку по "a"

s.split("\t") # разделяем строку по табуляции

"\t".join(list\_of\_strings) # объединяем

# СТАНДАРТНЫЙ ВВОД И ВЫВОД




# Давайте напишем свой wordcount

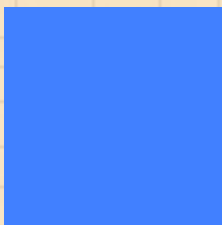
```
def wc():
```

```
    ...
```

```
    return res
```



sys.stdin, sys.stdout





# РАБОТА С ФАЙЛАМИ

```
with open("file.txt", "r") as f:  
    b = f.read()
```

```
with open("file2.txt", "r") as f:  
    for line in f:  
        print(line)
```



# ШАБЛОНИЗАЦИЯ И ФОРМАТИРОВАНИЕ

`s = u"ТеКсТ ДоМиКом"`

`s.lower()` # "текст домиком", есть так же `.upper()`

`"У Пети было {0} яблок".format(15)`

`"Что лучше - {0} ящиков пива или {1} ящиков водки?".format(30, 20)`  
"""

Взвесьте мне {0:.5f}

килограмм хурмы


""".format(21.35236172)

f-строки (Python 3.6+): `f"a + b = {a + b}"`


Старый формат:

printf-нотация: `"Жили в квартире %d веселых чижа" % 44`

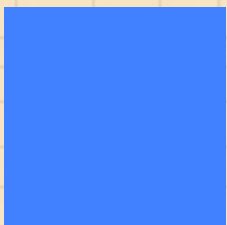
# ЧИТАЕМ ФАЙЛЫ С РАЗДЕЛИТЕЛЕМ



Почитаем файл *USDataLocalSites.csv* и посчитаем стандартное отклонение населения по всем штатам



`csv.reader, csv.DictReader`





# РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

# <https://docs.python.org/3/library/re.html>

# проверить, соответствует ли строка целиком

```
re.match(r"^\+7\d{10}$", "+78005553535").group(0)
```

```
a = """Пишите мне на адрес админ@суперхакер.рф или  
vasya@superhacker.me! Чмоки!"""
```

# найти первое вхождение

```
re.search(r"\w+@\w+\.\w{2,5}", a).group(0)
```

# найти все вхождения

```
re.findall(r"\w+@\w+\.\w{2,5}", a)
```

# лучше не компилировать в цикле

<http://regexpr.com>

# ОБРАБОТКА ОШИБОК

```
import traceback
```

```
try:
```

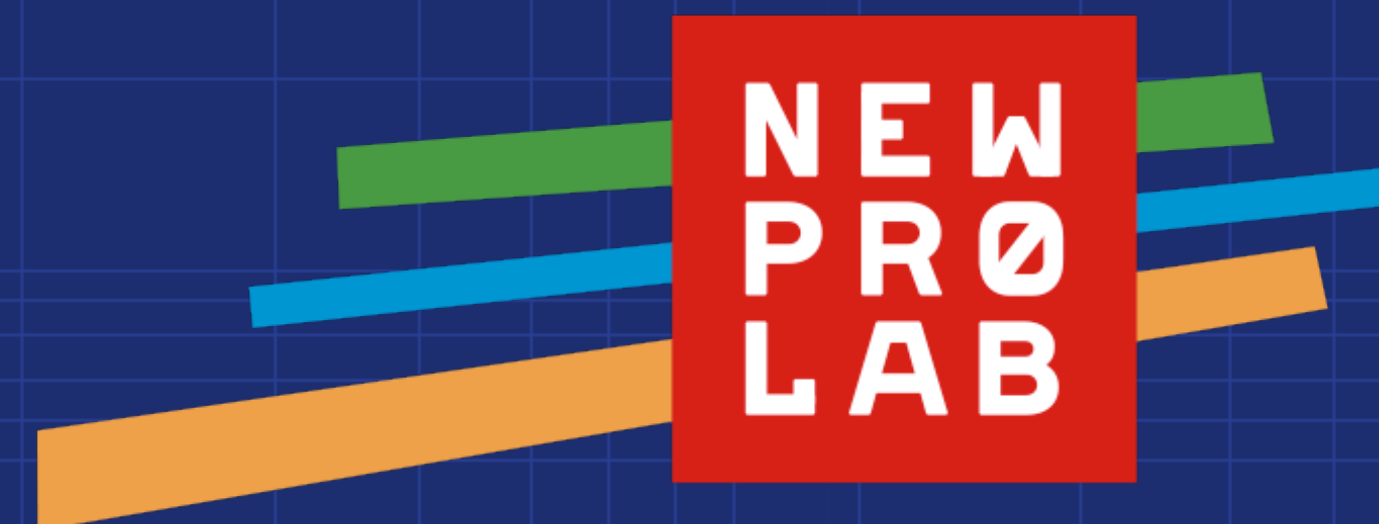
```
    1 / 0
```

```
except KeyError as exc:
```

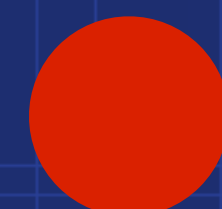
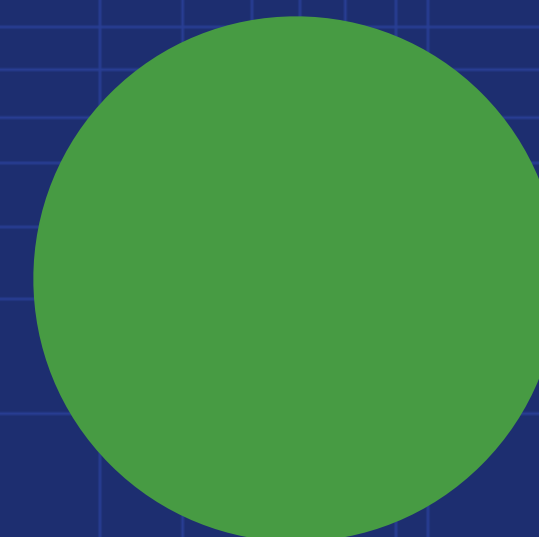
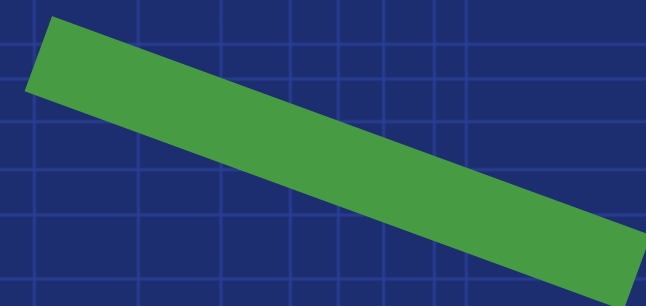
```
    print(traceback.format_exc())
```

```
except ZeroDivisionError as an_exc:
```

```
    print("bad luck")
```



# Веб и парсинг





# JSON

# <https://docs.python.org/3/library/json.html>

**import json**

# строка -> словарь

**d = json.loads('{"a": "foobar"}')**

# словарь -> строка


**s = json.dumps(d)**

При чтении из файла:

**with open("file.json", "r") as f:**

**d = json.load(f)**

# REQUESTS



# <http://docs.python-requests.org/en/latest/>

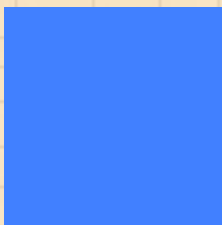

**import requests**

**r = requests.get('https://google.com/')**

**r.status\_code** # например, 200 или 500

**r.text** # http-ответ в текстовом формате

**r.json()** # http-ответ, если он в JSON-формате





# BEAUTIFUL SOUP

```
from bs4 import BeautifulSoup as BS
```



```
text = """<html>
  <body>
    <div class="container">
      <h2 align='center'>Some cool text</h2>
      <section>Some text here</section>
    </div>
    <ul>
      <li class="first">First choice</li>
      <li>Second choice</li>
    </ul>
  </body>
</html>"""
```



# BEAUTIFUL SOUP

```
soup = BS(text, "lxml")
# soup.find_all("li")[0].text
soup.find(
    "div",
    {"class": "container"}
).find(
    "h2",
    {"align": "center"}
).text
```

# БОЛЬШЕ ПАРСИНГА



1) Достать через Github API аватарку владельца репозитория python/cpython и сохранить на диск.

URL: <https://api.github.com/repos/python/cpython>

2) Давайте достанем текст превью к первой новости на сайте <http://python.org>

