

# Python

Куценко Никита

2023

1 / 65

CQG

# История

- В конце 70-х начале 80-х началась разработка ABC

Гвидо Ван Россум



<https://www.artima.com/articles/the-making-of-python>

# История

- В конце 70-х начале 80-х началась разработка ABC
- Присоединился к команде в 1983 году

Гвидо Ван Россум



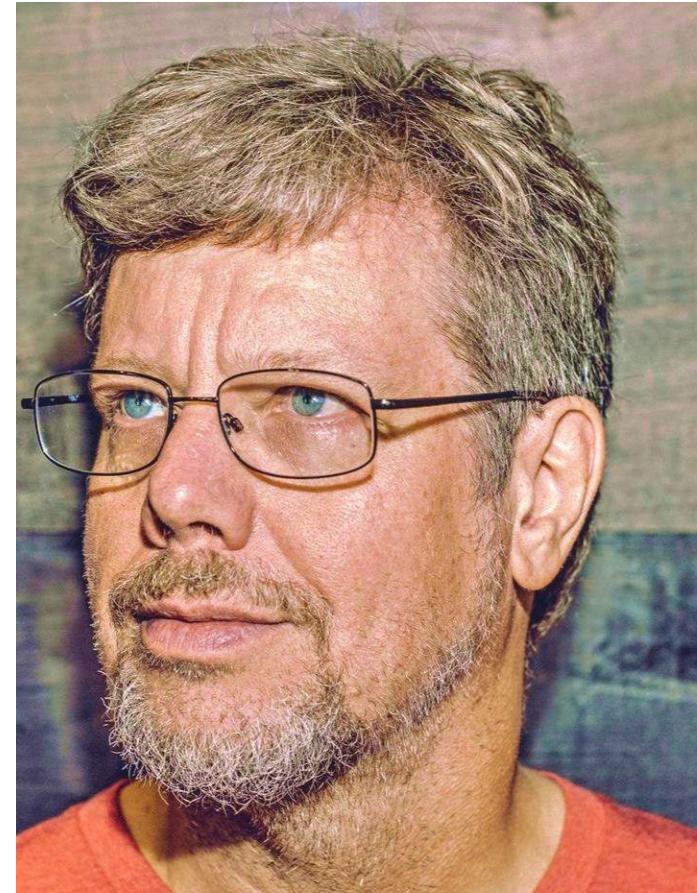
<https://www.artima.com/articles/the-making-of-python>

# История

- В конце 70-х начале 80-х началась разработка ABC
- Присоединился к команде в 1983 году
- 1986 - 1987 году прекращена разработка ABC

<https://www.artima.com/articles/the-making-of-python>

Гвидо Ван Россум



# История

- В конце 70-х начале 80-х началась разработка ABC
- Присоединился к команде в 1983 году
- 1986 - 1987 году прекращена разработка ABC
- 1986 году перешел в другой проект - в проект Amoeba

Гвидо Van Россум



<https://www.artima.com/articles/the-making-of-python>

# История

- В конце 70-х начале 80-х началась разработка ABC
- Присоединился к команде в 1983 году
- 1986 - 1987 году прекращена разработка ABC
- 1986 году перешел в другой проект - в проект Amoeba
- В рамках Amoeba приступил к разработке Python

<https://www.artima.com/articles/the-making-of-python>

Гвидо Ван Россум



# История

- В конце 70-х начале 80-х началась разработка ABC
- Присоединился к команде в 1983 году
- 1986 - 1987 году прекращена разработка ABC
- 1986 году перешел в другой проект - в проект Amoeba
- В рамках Amoeba приступил к разработке Python
- Python 1.0 вышел в 1994 году

Гвидо Ван Россум



<https://www.artima.com/articles/the-making-of-python>

# Python Enhancement Proposals

«*PEP расшифровывается как предложение по улучшению Python. PEP - это проектный документ, предоставляющий информацию сообществу Python или описывающий новую функцию для Python или его процессов или среды. PEP должен содержать краткую техническую спецификацию функции и обоснование для этой функции.*» - *PEP1*

<https://peps.python.org>

# Python Enhancement Proposals

«*PEP расшифровывается как предложение по улучшению Python. PEP - это проектный документ, предоставляющий информацию сообществу Python или описывающий новую функцию для Python или его процессов или среды. PEP должен содержать краткую техническую спецификацию функции и обоснование для этой функции.*» - *PEP1*

- PEP8 – Руководство по стилю для кода Python
- PEP484 – Подсказки типов
- PEP498 – f-строки
- PEP693 – График выпуска Python 3.12

<https://peps.python.org>

# Python Enhancement Proposals

## PEP 8 – Style Guide for Python Code

**Author:** Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>

**Status:** Active

**Type:** Process

**Created:** 05-Jul-2001

**Post-History:** 05-Jul-2001, 01-Aug-2013

---

### ► Table of Contents

## Introduction

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing [style guidelines for the C code in the C implementation of Python](#).

This document and [PEP 257](#) (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide [2].

<https://peps.python.org>

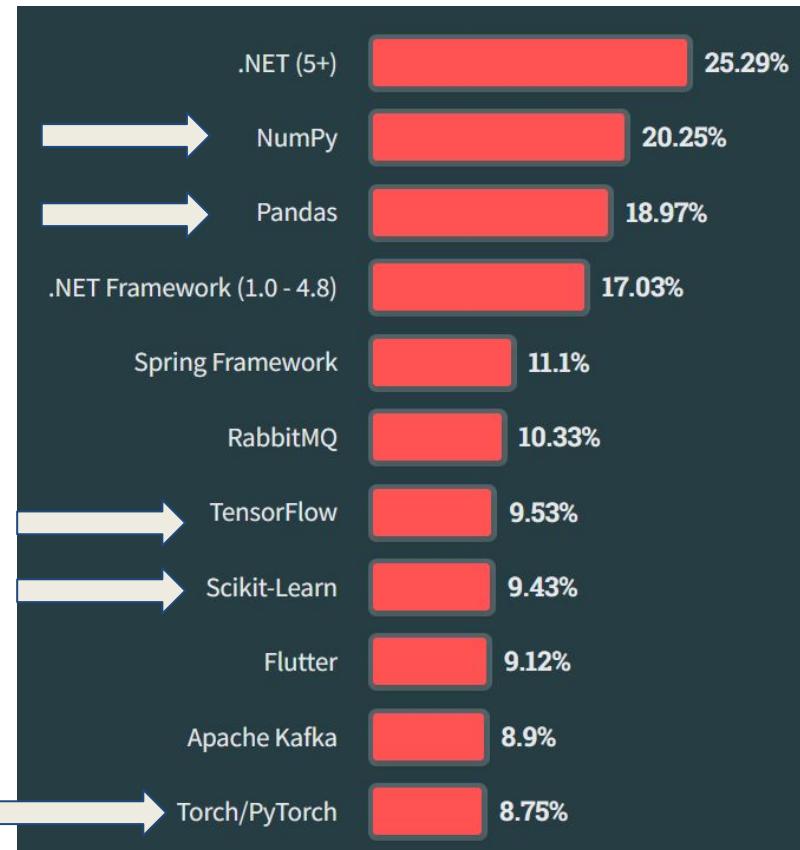
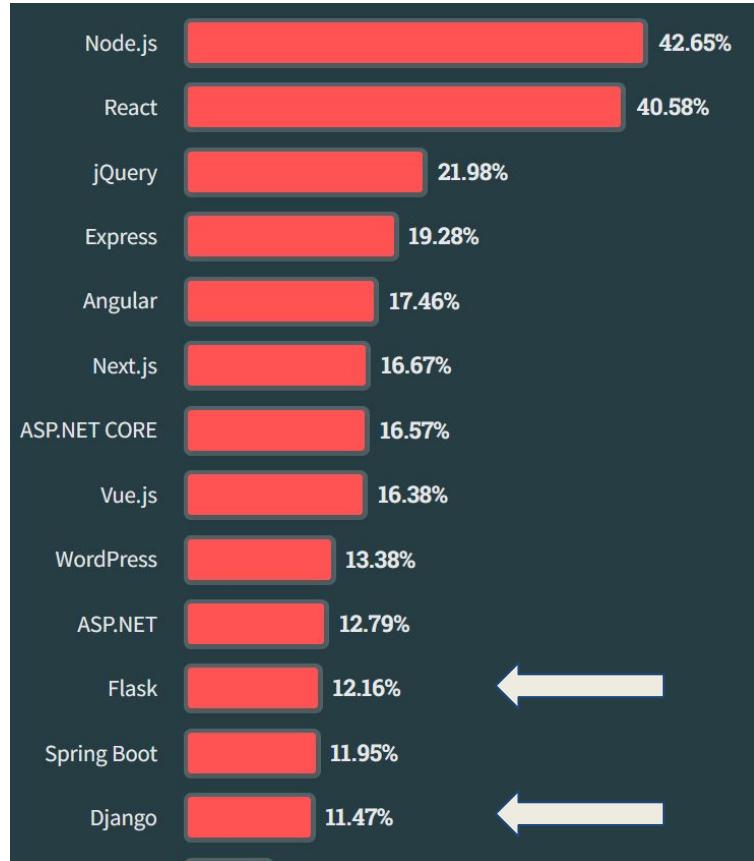
# Популярность

- Используют 49% разработчиков.
- 2/3 тех, кто пользовались, остались довольны языком.



<https://survey.stackoverflow.co/2023/>

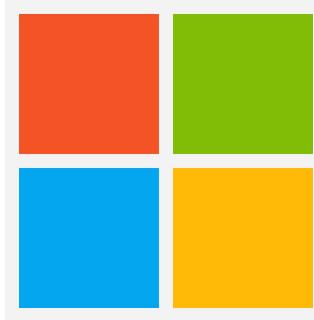
# Популярность



<https://survey.stackoverflow.co/2023/>

# Кто использует Python?

- Google
- Dropbox
- Яндекс
- CQG
- и тд.

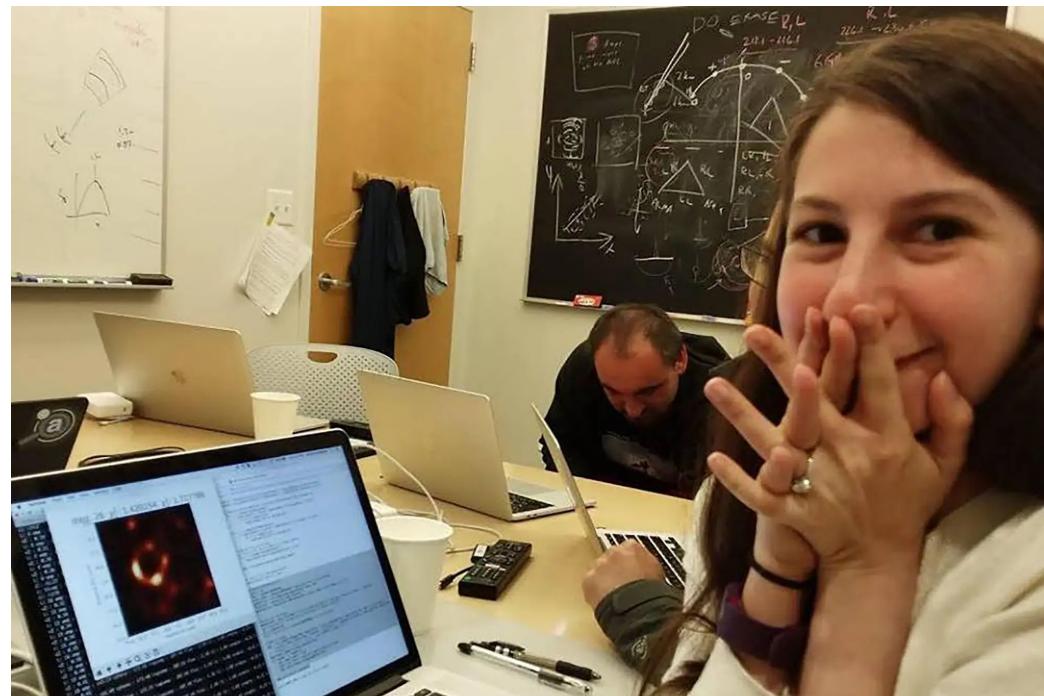


# Кто ещё использует Python?

В школах и ВУЗах Python часто используется как язык для обучения программированию, за счет своего простого синтаксиса.

Физики и математики используют Python для сложных вычислений.

Так-же Python является главным языком в машинном обучении и анализе данных.



# Переменные

```
boolean = True                      # bool
boolean = False                     # bool
number = 123                        # int
string = "hello, world!"            # str
float_number = 1.23                  # float
array = [1, 2.0, "three"]           # list
tuple = (1, 2.0, "three")           # tuple
dictionary = {"key1": "value1", "key2": "value2"} # dict
# комментарий
```

and, or, not

left + right  
left \* right  
left - right  
left \*\* right  
<<, >>, ...

mutable:  
list, dict

immutable:  
bool, int, float, tuple

# Ветвление и циклы

```
if password == "admin" and username == "admin":  
    print("You are the admin")  
else:  
    print("You are not the admin")
```

```
array = [1, 2.0, "three"]  
for value in array:  
    print(value)  
  
# 1  
# 2.0  
# three
```

```
for i in range(7, 11):  
    print(i)  
  
# 7  
# 8  
# 9  
# 10
```

<https://peps.python.org/pep-0008/#indentation>

# ФУНКЦИИ

```
def add(l, r):  
    return l + r  
  
def multiply(l: int, r: int) -> int:  
    return l * r
```

```
add(1, 3) # 4  
multiply(3, 4) # 12  
multiply("hi!", 5) # "hi!hi!hi!hi!hi!"
```

<https://peps.python.org/pep-0484/>

<https://mypy-lang.org/>

# Классы

```
class Rectangle:  
    def __init__(self, width, height):  
        self.width = width  
        self.height = height  
  
    def area(self):  
        return self.width * self.height  
  
rect = Rectangle(15, 15)  
  
print(rect.area()) # 225
```

# Менеджер контекста

```
class Context:  
    def __enter__(self):  
        print("__enter__")  
  
    def method(self):  
        return "In method"  
  
    def __exit__(self, *args):  
        print("__exit__")  
  
with Context() as c:  
    print(c.method())  
  
# __enter__  
# In method  
# __exit__
```

# List comprehension и работа со строками

```
list = [i ** 2 for i in range(1, 11)]
print(list) # [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
var = "World"
s1 = "Hello {}".format(var) # Hello World
s2 = "Hello {}" % (var, ) # Hello World
s3 = f"Hello {var}" # Hello World
```

# Декораторы

```
def say_hello_before_call(f):
    def __f():
        print("Hello!")
        f()
    return __f

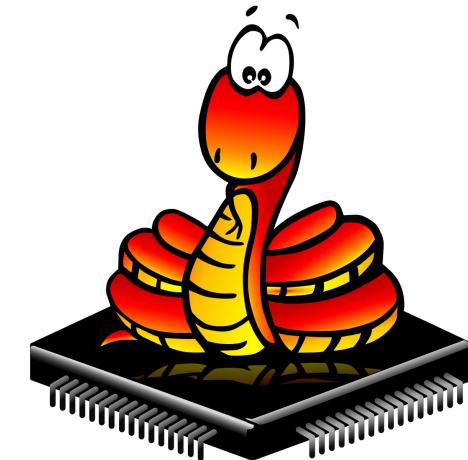
@say_hello_before_call
def hard_work():
    print("Doing a lot of work")

hard_work()

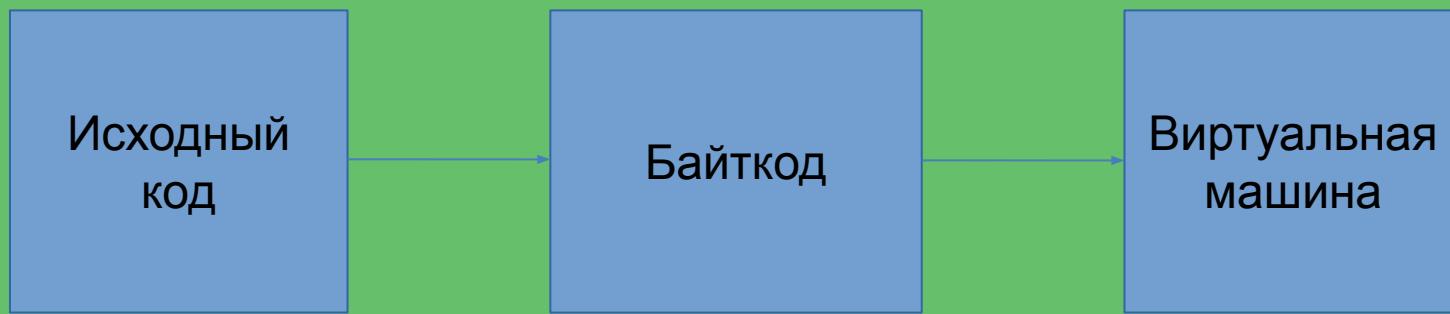
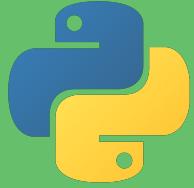
# Hello!
# Doing a lot of work
```

# Реализации

Iron  
Python



# Python - интерпретируемый



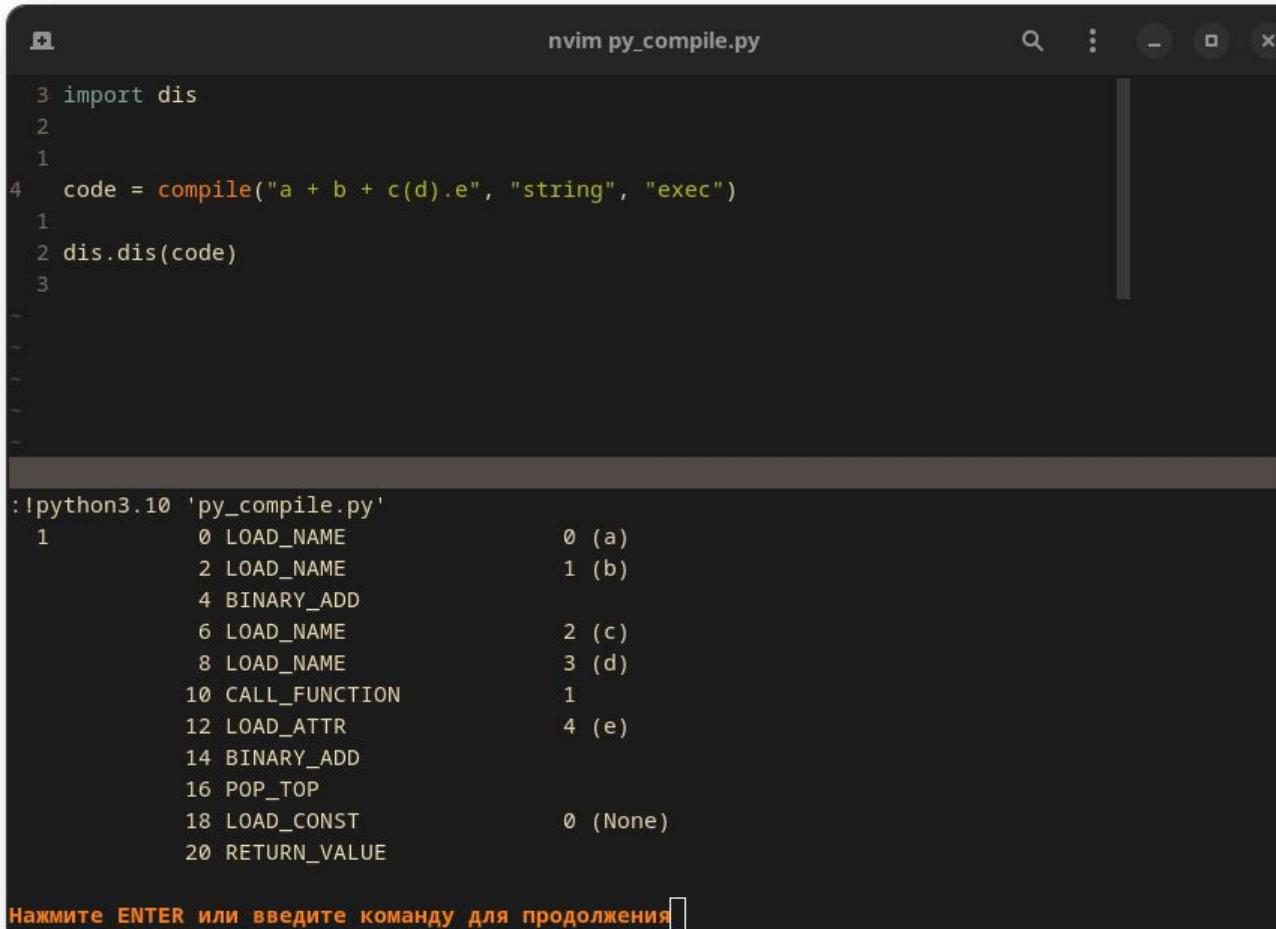
# Пример байткода

```
x = 2023
y = "Hello world"
print(y, x)

y = 1
z = x + y
```

3	0 LOAD_CONST 2 STORE_FAST	1 (2023) 0 (x)
4	4 LOAD_CONST 6 STORE_FAST	2 ('Hello world') 1 (y)
6	8 LOAD_GLOBAL 10 LOAD_FAST 12 LOAD_FAST 14 CALL_FUNCTION 16 POP_TOP	0 (print) 1 (y) 0 (x) 2
8	18 LOAD_CONST 20 STORE_FAST	3 (1) 1 (y)
9	22 LOAD_FAST 24 LOAD_FAST 26 BINARY_ADD 28 STORE_FAST	0 (x) 1 (y) 2 (z)

# Компилируется != Исполняется

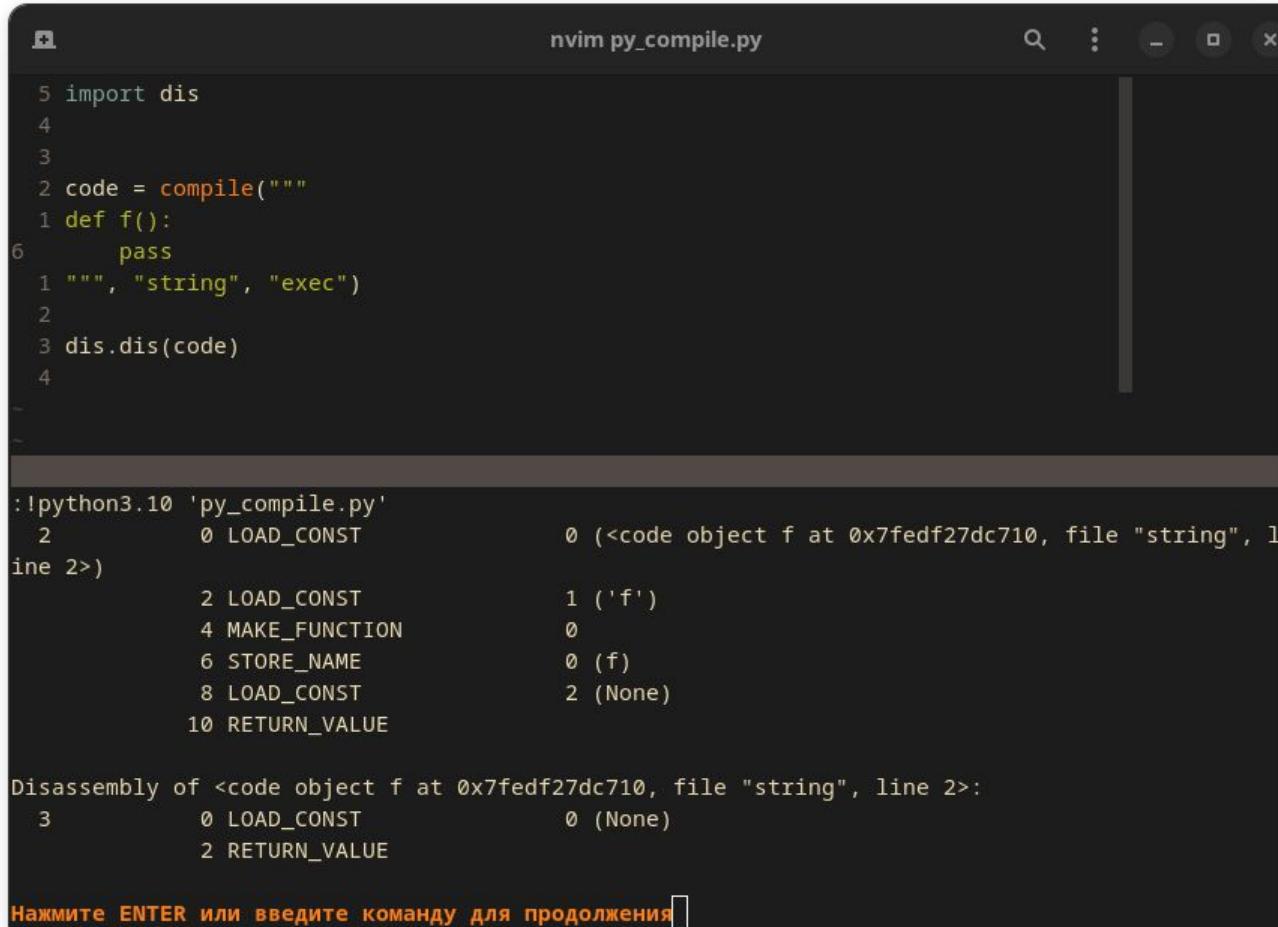


```
nvim py_compile.py
3 import dis
2
1
4 code = compile("a + b + c(d).e", "string", "exec")
1
2 dis.dis(code)
3

:!:python3.10 'py_compile.py'
 1      0 LOAD_NAME              0 (a)
 2 LOAD_NAME              1 (b)
 4 BINARY_ADD
 6 LOAD_NAME              2 (c)
 8 LOAD_NAME              3 (d)
10 CALL_FUNCTION          1
12 LOAD_ATTR               4 (e)
14 BINARY_ADD
16 POP_TOP
18 LOAD_CONST              0 (None)
20 RETURN_VALUE

Нажмите ENTER или введите команду для продолжения
```

# Функции создаются в рантайме



```
nvim py_compile.py

5 import dis
6
7
8 code = compile("""
9 def f():
10     pass
11     """
12     , "string", "exec")
13
14 dis.dis(code)
15

~
~

:!python3.10 'py_compile.py'
 2          0 LOAD_CONST      0 (<code object f at 0x7fedf27dc710, file "string", line 2>)
 2          2 LOAD_CONST      1 ('f')
 4 MAKE_FUNCTION
 6 STORE_NAME       0 (f)
 8 LOAD_CONST      2 (None)
10 RETURN_VALUE

Disassembly of <code object f at 0x7fedf27dc710, file "string", line 2>:
 3          0 LOAD_CONST      0 (None)
 2 RETURN_VALUE

Нажмите ENTER или введите команду для продолжения
```

# Интерпретатор - плюсы и минусы

## Плюсы

- Кроссплатформенность
- Гибкость
- Рефлексия и интроспекция
- Динамическая типизация
- Модификация программы во время исполнения

## Минусы

- Малая скорость выполнения
- Большая требовательность к ресурсам
- Выше требования к корректности написанных программ

# Все объект

```
nvim py_compile.py
```

```
1
1 def f():
2     ...
3
4 class A:
5     ...
6
7 print(type(1))
8 print(type(f))
9 print(type(A))
10 print(type(print))
11 print(type(type))

~
~
~
```

```
:!python3.10 'py_compile.py'
<class 'int'>
<class 'function'>
<class 'type'>
<class 'builtin_function_or_method'>
<class 'type'>
```

Нажмите ENTER или введите команду для продолжения

# Объекты и ссылки

```
a = []
b = a

a.append(1)

print(b) # [1]

print(a == b) # True
print(a is b) # True
```



```
std::vector<int> a;
auto b = a;

a.push_back(1);

print(b); # []

print(a == b); # false
print(&a == &b); # false
```



# Объекты и ссылки

```
a = []
b = a

a.append(1)

print(b) # [1]

print(a == b) # True
print(a is b) # True
```



```
std::vector<int> a;
auto& b = a;

a.push_back(1);

print(b); # [1]

print(a == b); # true
print(&a == &b); # true
```



# Объекты и ссылки

```
a = 12
b = a

a += 7

print(b) # 12

print(a == b) # False
print(a is b) # False
```



```
int a = 12;
int b = a;

a += 7;

print(b); # 12

print(a == b); # false
print(&a == &b); # false
```



# Объекты и ссылки

```
a = 12
b = a

a += 7

print(b) # 12

print(a == b) # False
print(a is b) # False
```



```
int a = 12;
int& b = a;

a += 7;

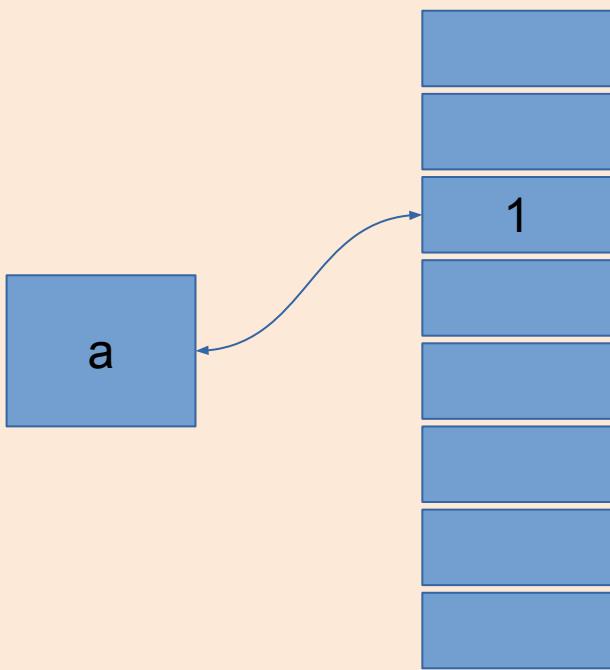
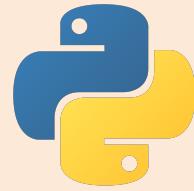
print(b); # 19

print(a == b); # true
print(&a == &b); # true
```

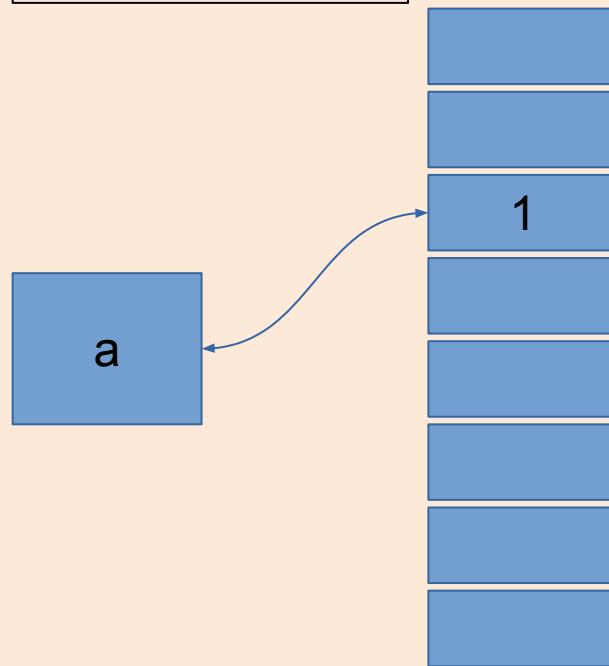


# Объекты и ссылки

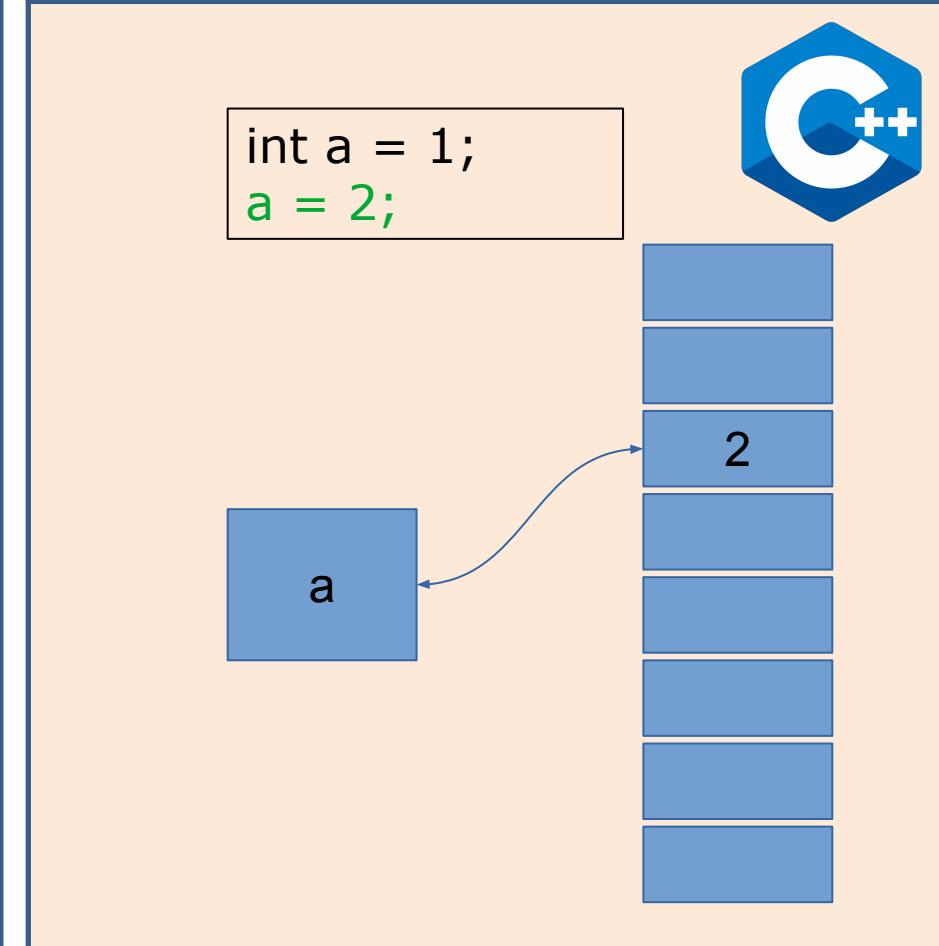
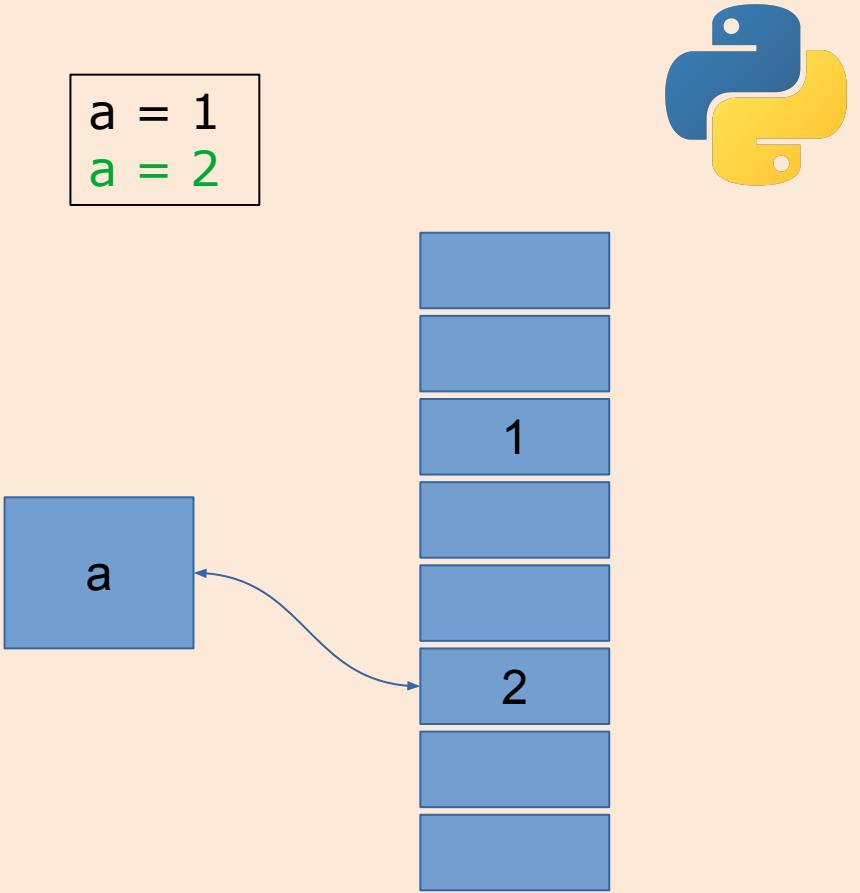
```
a = 1  
a = 2
```



```
int a = 1;  
a = 2;
```

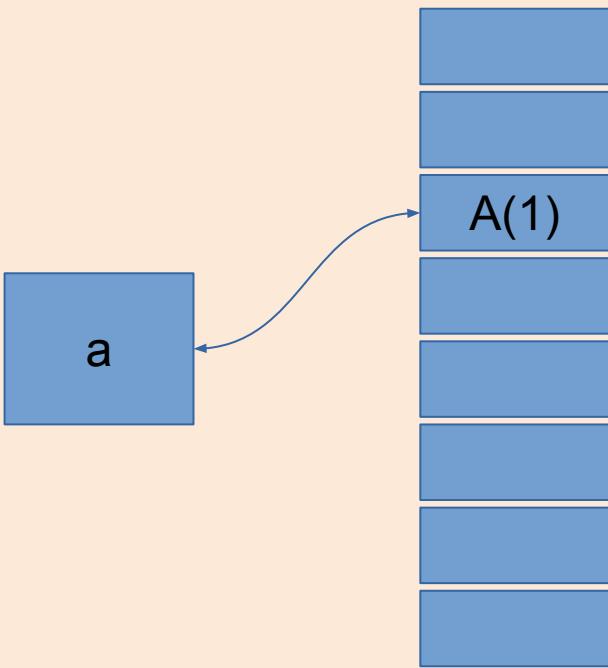
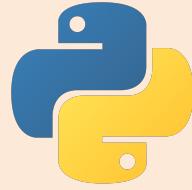


# Объекты и ссылки

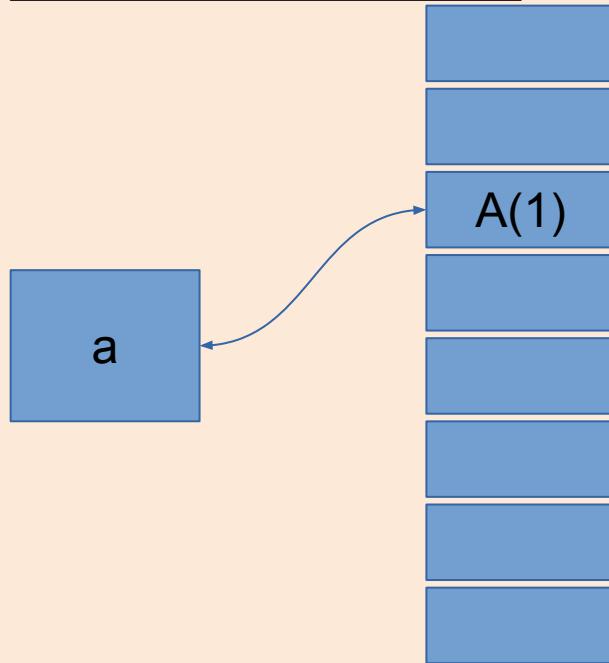
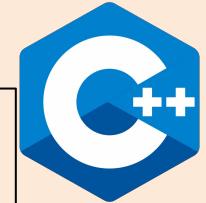


# Сборка мусора

```
a = A(1)  
a = A(2)
```

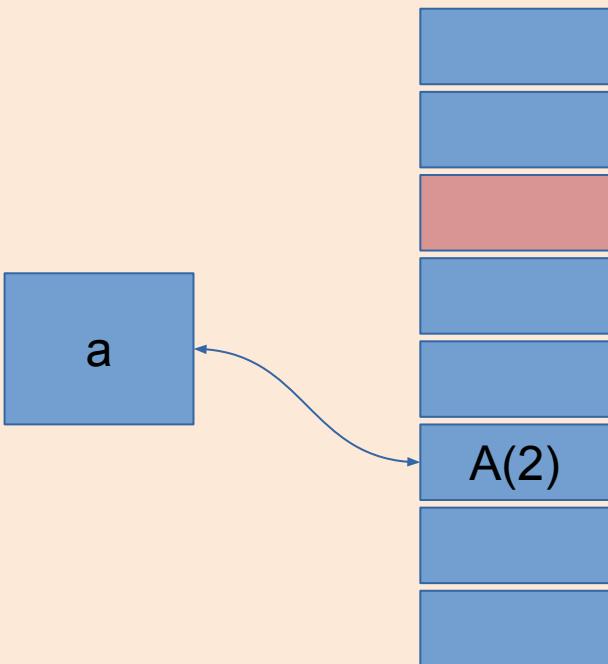
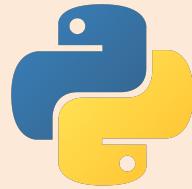


```
A* a = new A(1);  
a = new A(2);
```

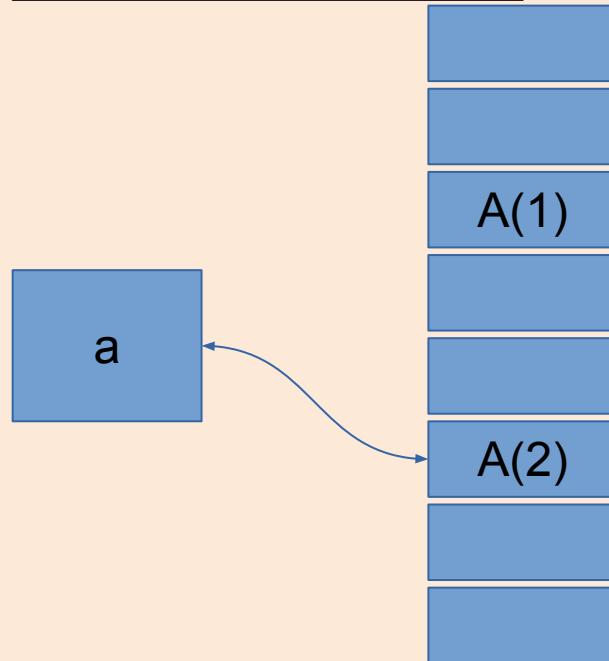
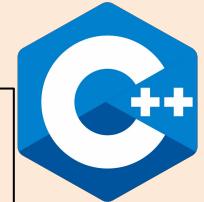


# Сборка мусора

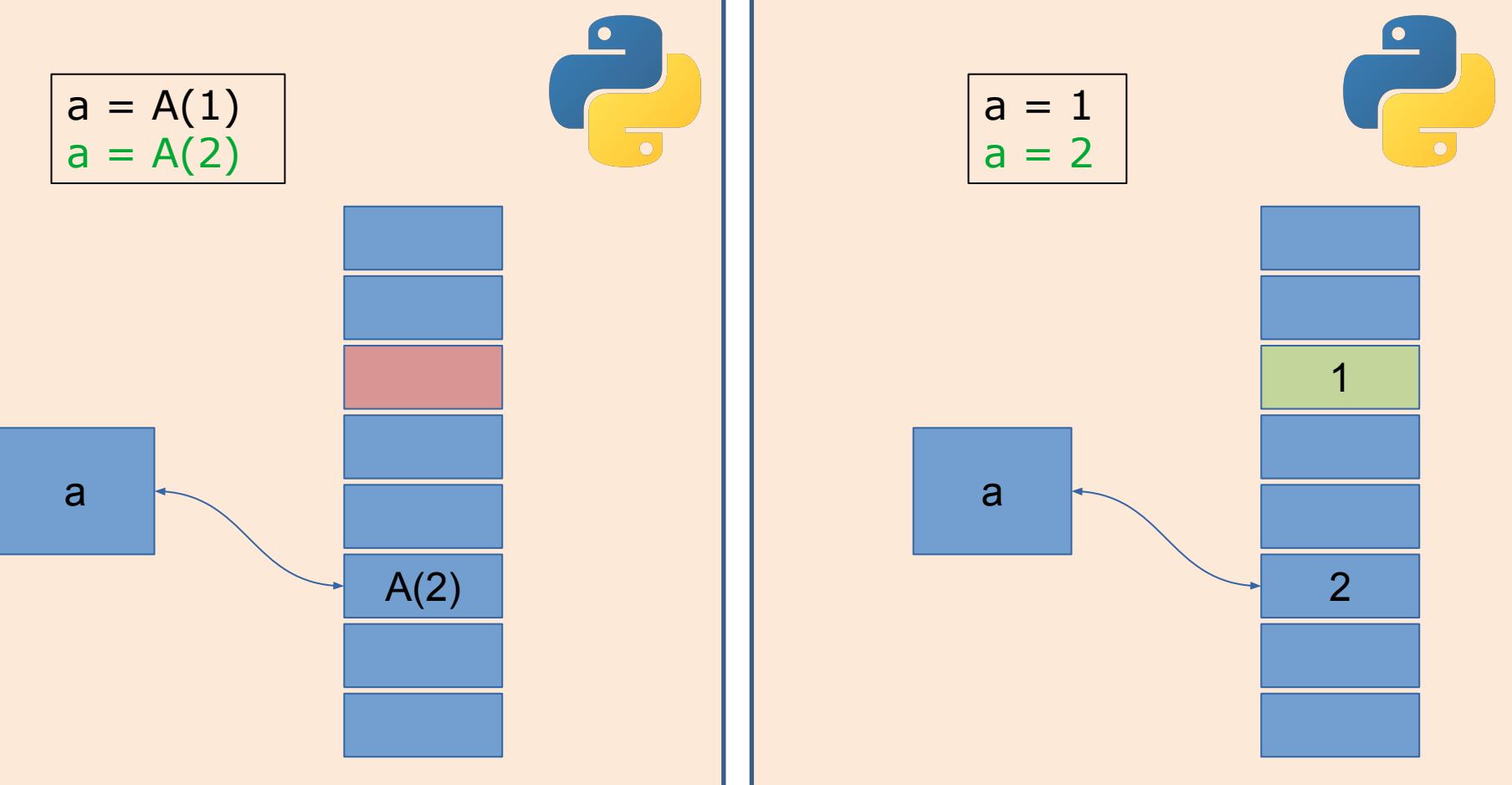
```
a = A(1)  
a = A(2)
```



```
A* a = new A(1);  
a = new A(2);
```



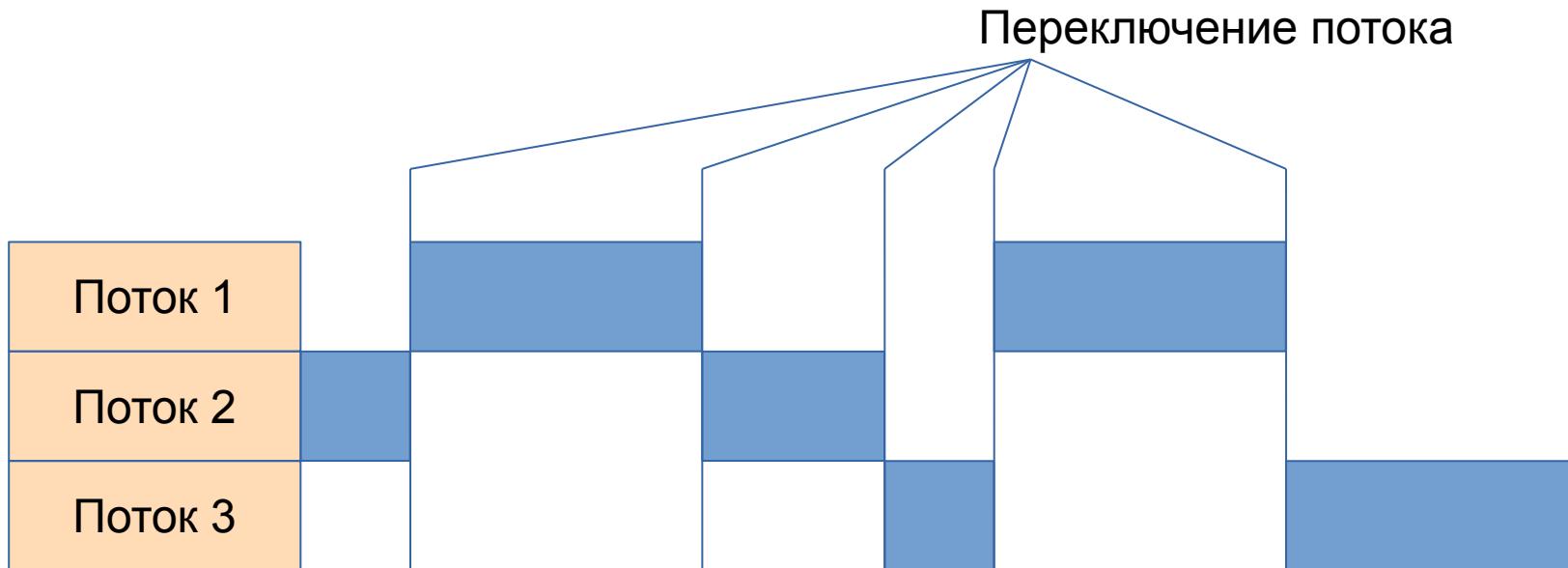
# Сборка мусора



<https://stackoverflow.com/questions/57296092/why-cpython-pre-allocates-some-integer-numbers>

# GIL

## Глобальный блокировщик интерпретатора



<https://wiki.python.org/moin/GlobalInterpreterLock>

# GIL - плюсы и минусы

## Плюсы

- Защищает интерпретатор от ошибок многопоточности

## Минусы

- Не защищает ваш код от ошибок многопоточности
- Потоки выполняются последовательно

# Сравнение с C++

Сравнивать будем между собой Python и C++

Что будем сравнивать:

- Стандартную библиотеку
- Сторонние библиотеки
- Скорость работы
- Потребление памяти
- Скорость разработки
- Как установить?

# Стандартная библиотека

	Python	C++
GUI		
Алгоритмы и контейнеры		
Сериализация Десериализация		
Работа с сетью		
Ввод/Выход		
Работа с ОС		
Многопоточность		
Тестирование		

and more...

# Сторонние библиотеки

- Машинное обучение (tensorflow, jax, pytorch, ...)
- Бэкенд (tornado, fastapi, Django, ...)
- Математика (numpy, scipy, sympy, numba, ...)
- Игры (pygame, renpy, ...)
- GUI (pygtk, pyqt, ...)
- Работа с БД (sqlalchemy, ...)
- Web (requests, ...)
- и тд

# Скорость работы!

## Фрактал Мандельброта

Точки на координатной плоскости ставим в соответствие количество итераций, за которое ряд расходится

$$z_{n+1} = z_n^2 + z_0$$

где  $z = x + iy$ .

Посчитаем для массива 512x512

## Подсчет слов

Считается количество вхождений слов в текст. Например „Подсчет слов: Считается количество вхождений слов в текст“

Подсчет → 1

Слов → 2

Считается → 1

Количество → 1

Вхождений → 1

В → 1

Текст → 1

# Скорость работы!

	Python	C++
Фрактал	1.5 секунды	0.12 секунд
Слова	8.35 секунд	4.61 секунд

# Скорость работы!

	Python (+numba)	C++
Фрактал	0.12 секунд	0.12 секунд
Слова	8.35 секунд	4.61 секунд

# Скорость работы!

```
def mandelbrot(z: complex) -> int:
    z0 = z
    k = 0
    while abs(z) < 2. and k < 200:
        k += 1
        z = z**2 + z0
    return k

def work():
    W, H = 512, 512
    data = [[0] * W for _ in range(H)]

    for i in range(H):
        for j in range(W):
            x, y = 2.* (2.*i-H)/H, 2.* (2.*j-W)/W
            data[i][j] = mandelbrot(x + 1j*y)

work()
```

# Скорость работы!

```
import numba

@numba.njit
def mandelbrot(z: complex) -> int:
    z0 = z
    k = 0
    while abs(z) < 2. and k < 200:
        k += 1
        z = z**2 + z0
    return k

@numba.njit
def work():
    W, H = 512, 512
    data = [[0] * W for _ in range(H)]

    for i in range(H):
        for j in range(W):
            x, y = 2.* (2.*i-H)/H, 2.* (2.*j-W)/W
            data[i][j] = mandelbrot(x + 1j*y)

work()
```

<https://numba.pydata.org/>

# Память

	Python	Python (+numba)	C++
Фрактал	3.0 (5.12) MB	17.4 (93.89) MB	1.08 (1.08) MB
Слова	4.7 (874.12) MB	4.7 (874.12) MB	4.0 (4.56) MB

# Скорость разработки

```
#include <iostream>
#include <unordered_map>

int main() {
    std::unordered_map<std::string, int> counts;

    std::ifstream file("big_file.txt");
    std::string word;

    while (file >> word) { ++counts[word]; }

    return 0;
}
```

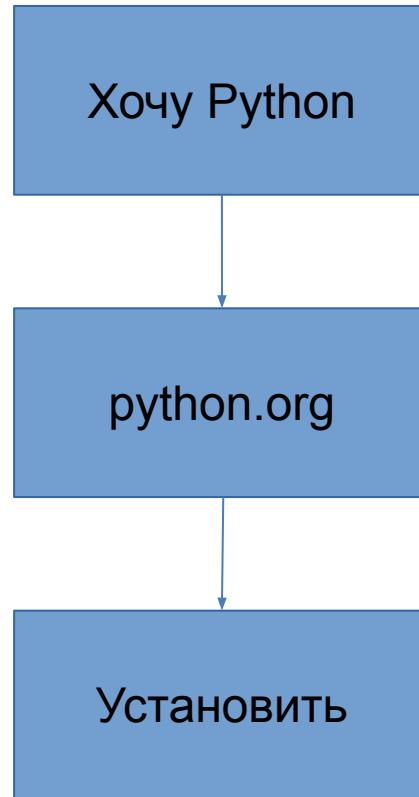


```
from collections import Counter

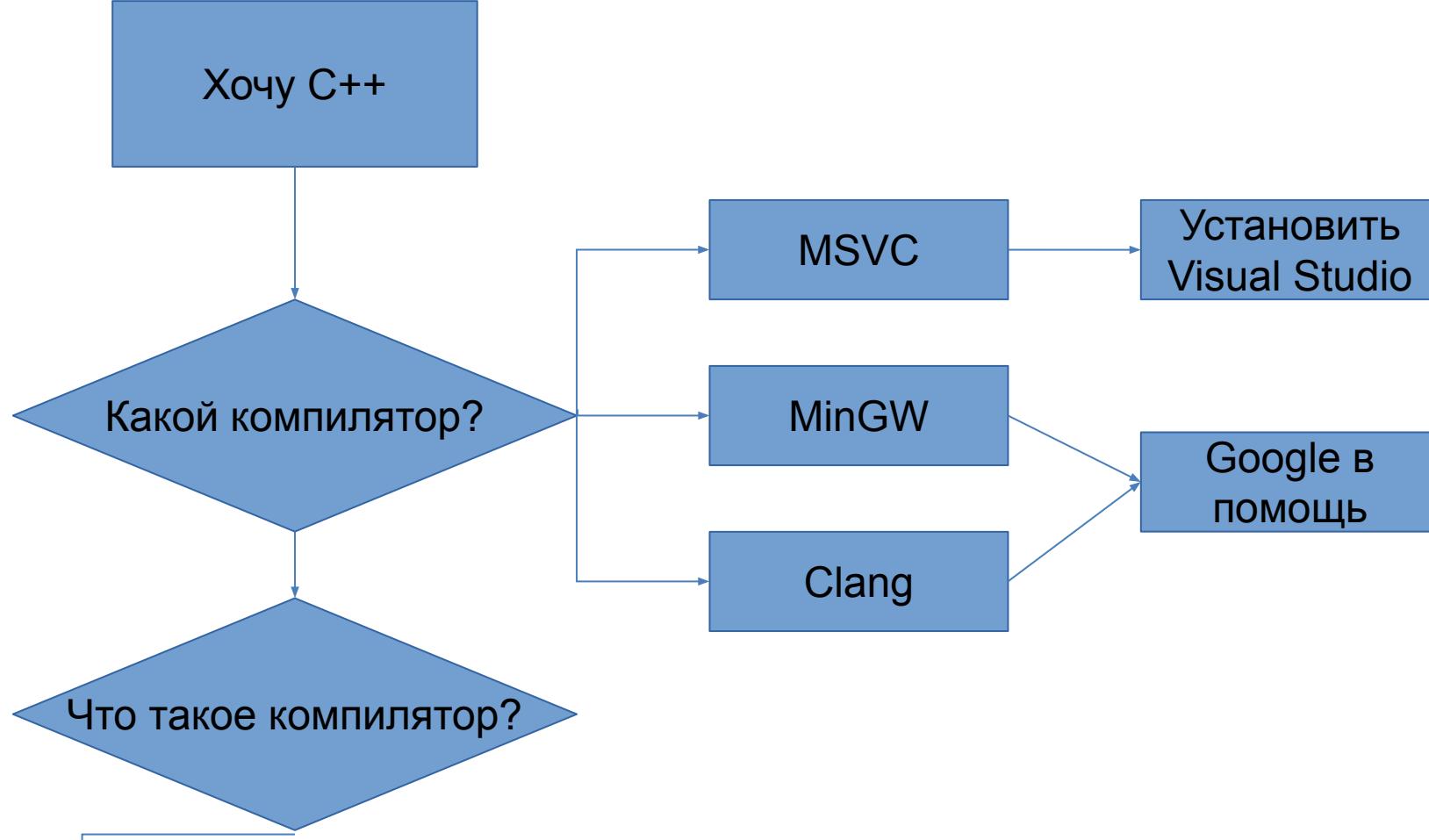
with open('big_file.txt', 'r') as file:
    counts = Counter((w for line in file for w in line.split()))
```



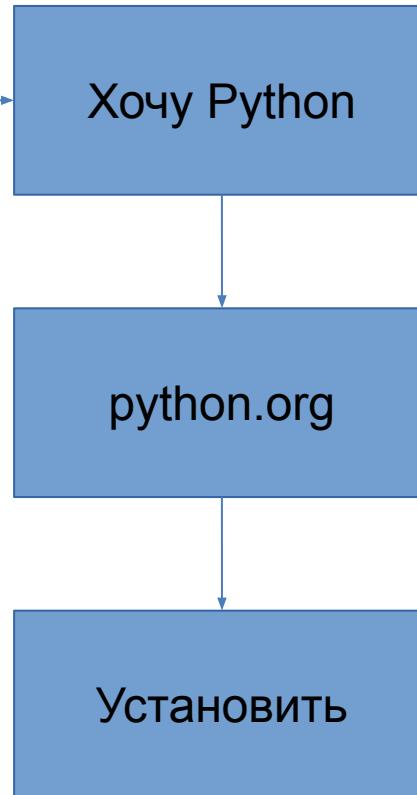
# Как установить Python?



# Как установить C++?



# Как установить Python?



# Выводы сравнения

Плюсы Python по сравнению с C++

- Большая стандартная библиотека
- Простота в разработке
- Легкий старт

Минусы Python по сравнению с C++

- Медленный
- Требовательный к ресурсам

# Pip

Позволяет легко устанавливать сторонние Python пакеты

```
pip install scipy
Defaulting to user installation because normal site-packages is not writeable
Collecting scipy
  Downloading scipy-1.9.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (33.7 MB)
   ━━━━━━━━━━━━━━━━ 33.7/33.7 MB 46.0 MB/s eta 0:00:00
Requirement already satisfied: numpy<1.26.0,>=1.18.5 in ./local/lib/python3.10/site-packages (from scipy) (1.23.4)
Installing collected packages: scipy
Successfully installed scipy-1.9.3
```

<https://pip.pypa.io/en/stable/>

# requirements.txt

Позволяет установить список зависимостей

The screenshot shows a terminal window with the following session:

```
nikita@nikitapc:~/code/lecture/env
~/c/l/env pip freeze > requirements.txt
~/c/l/env cat requirements.txt
numpy==1.23.5
~/c/l/env pip install -r requirements.txt
Requirement already satisfied: numpy==1.23.5 in ./lib/python3.10/site-packages (
from -r requirements.txt (line 1)) (1.23.5)

[notice] A new release of pip available: 22.2.2 -> 22.3.1
[notice] To update, run: pip install --upgrade pip
~/c/l/env deactivate
~/code/lecture/env
```

The terminal window has a dark background with light-colored text. It shows the user's path as `nikita@nikitapc:~/code/lecture/env`. The user runs `pip freeze > requirements.txt` to generate a requirements file. Then, they use `cat requirements.txt` to view its contents, which show the dependency `numpy==1.23.5`. Finally, they run `pip install -r requirements.txt` to install the package. The terminal also displays notices about a new pip release.

<https://pip.pypa.io/en/stable/reference/requirements-file-format/>

# Venv

Инструмент venv поддерживает создание легковесных “виртуальных окружений”, каждое из которых со своим собственным независимым набором Python пакетов.

```
~/c/c/python/environment python -m venv new_env
~/c/c/python/environment ls
new_env
~/c/c/python/environment cd new_env
~/c/c/p/e/new_env ls
bin include lib lib64 pyvenv.cfg
~/c/c/p/e/new_env
```

The screenshot shows a terminal window with a dark background and light-colored text. It displays the command `python -m venv new\_env` followed by the output `new\_env`. Then, it shows the command `cd new\_env` followed by the output `bin include lib lib64 pyvenv.cfg`. To the right of the terminal window, there are four green checkmark icons indicating successful execution of the commands.

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

# Venv

```
nikita@nikitapc:~/code/lecture/env python -m venv .
nikita@nikitapc:~/code/lecture/env source ./bin/activate
nikita@nikitapc:~/c/l/env pip list
Package      Version
-----
pip          22.2.2
setuptools   63.2.0

[notice] A new release of pip available: 22.2.2 -> 22.3.1
[notice] To update, run: pip install --upgrade pip
nikita@nikitapc:~/c/l/env pip install numpy
Collecting numpy
  Downloading numpy-1.23.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.1 MB)
    17.1/17.1 MB 7.6 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.23.5

[notice] A new release of pip available: 22.2.2 -> 22.3.1
[notice] To update, run: pip install --upgrade pip
nikita@nikitapc:~/c/l/env pip freeze
numpy==1.23.5
nikita@nikitapc:~/c/l/env
```

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

# Примеры использования

- Замена MATLAB
- Поиск неиспользуемых констант
- Вычисление сложных производных
- Скрипт, чтобы мышь трясти



1.  $c' = 0, c = \text{const}$
2.  $(x^n)' = nx^{n-1}$
3.  $(a^x)' = a^x \cdot \ln a$
4.  $(e^x)' = e^x$
5.  $(\log_a x)' = \frac{1}{x \ln a}$
6.  $(\ln x)' = \frac{1}{x}$
7.  $(\sin x)' = \cos x$
8.  $(\cos x)' = -\sin x$
9.  $(\sqrt{x})' = \frac{1}{2\sqrt{x}}$
10.  $(\operatorname{tg} x)' = \frac{1}{\cos^2 x}$
11.  $(\operatorname{ctg} x)' = -\frac{1}{\sin^2 x}$
12.  $(\arcsin x)' = \frac{1}{\sqrt{1-x^2}}$
13.  $(\arccos x)' = -\frac{1}{\sqrt{1-x^2}}$
14.  $(\operatorname{arctg} x)' = \frac{1}{1+x^2}$
15.  $(\operatorname{arcctg} x)' = -\frac{1}{1+x^2}$
16.  $(\operatorname{sh} x)' = \operatorname{ch} x$
17.  $(\operatorname{ch} x)' = \operatorname{sh} x$
18.  $(\operatorname{th} x)' = \frac{1}{\operatorname{ch}^2 x}$
19.  $(\operatorname{th} x)' = -\frac{1}{\operatorname{sh}^2 x}$

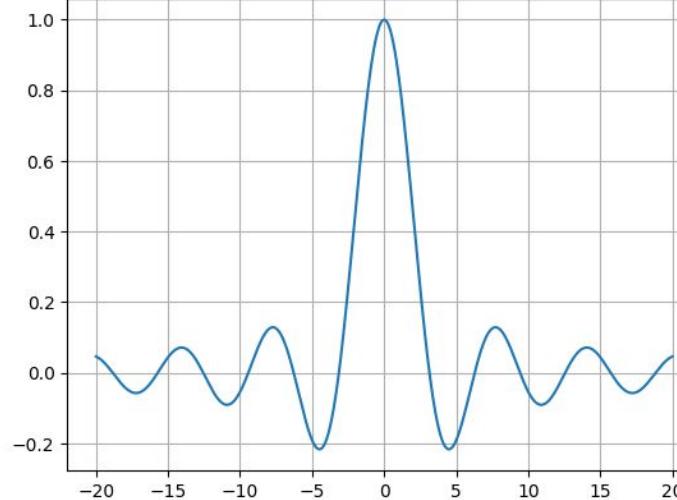
# Jupyter + numpy + matplotlib

jupyter Untitled Last Checkpoint: 3 minutes ago (unsaved changes)

In [1]: `import numpy as np  
import matplotlib.pyplot as plt`

In [2]: `x = np.linspace(-20, 20, 1000)  
y = np.sin(x) / x`

In [3]: `plt.plot(x, y)  
plt.grid()`



<https://jupyter.org/>

<https://numpy.org/>

<https://matplotlib.org/>

# Поиск ненеиспользуемых констант

```
1 import re
2
3 import requests as rq
4 from bs4 import BeautifulSoup
5
6 file = open("C:/project/.../code.cpp").read()
7
8 for name in re.findall("#define\\s+(\\w+)\\s+\\d+", file):
9     response = rq.get(f"search.in_code.cqg/search?full={name}")
10    html = BeautifulSoup(response.text, "parse.html")
11    if len(html.find_all("tag")) > 1:
12        print(name)
```

<https://requests.readthedocs.io/en/latest/>

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

# Сложная символьная математика

```
1 import sympy as sp
2
3 x = sp.symbols("x")
4 y = sp.sin(x**2) / x
5
6 der4 = sp.diff(y, x, 4)
7
8 sp.print_latex(der4.simplify())
9
```

$$16x^3 \sin(x^2) - 16x \cos(x^2) - \frac{12 \sin(x^2)}{x} - \frac{24 \cos(x^2)}{x^3} + \frac{24 \sin(x^2)}{x^5}$$

<https://www.sympy.org/en/index.html>

# Тряска мыши

```
1 import time
2 import win32api
3
4 right = True
5 while True:
6     x, y = win32api.GetCursorPos()
7     win32api.SetCursorPos((x + 1 if right else -1, y))
8     right = not right
9     time.sleep(1.)
```

<http://timgolden.me.uk/pywin32-docs/>

# Где уместен Python?

- Бэкэнд

Python хорошо применять для быстрого старта разработки приложения. Если вы понимаете, что нагрузки будут не высокими, и вы не сильно ограничены ресурсами, то Python ваш выбор.

- Математика и машинное обучение

Да, численный счет не являются коньком Python, но тут высокая производительность от него и не требуется, большое число математических библиотек написаны на C/C++, а Python'у остаётся только брать их и использовать.

- Автоматизация

Python прекрасно себя показывает в качестве языка автоматизации и написания небольших утилит. Обычно такие задачи не требуют больших вычислений.

# Ссылки

Где скачать - <https://www.python.org/>

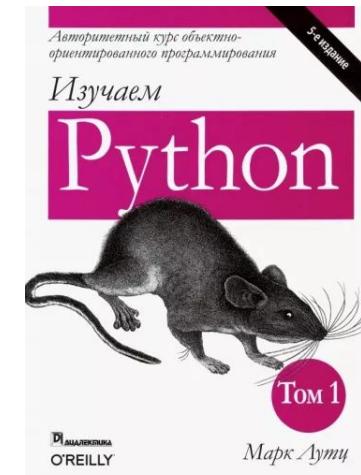
The Python Tutorial - <https://docs.python.org/3/tutorial/index.html>

Самоучитель - <https://pythonworld.ru/samouchitel-python>

Про CPython - <https://www.youtube.com/watch?v=PxlqLgitQ5Y>

Про GIL - <https://www.youtube.com/watch?v=AWX4JnAnjBE>

Большой сайт про Python - <https://realpython.com/>



Марк Лутц

CQG

# Q&A

