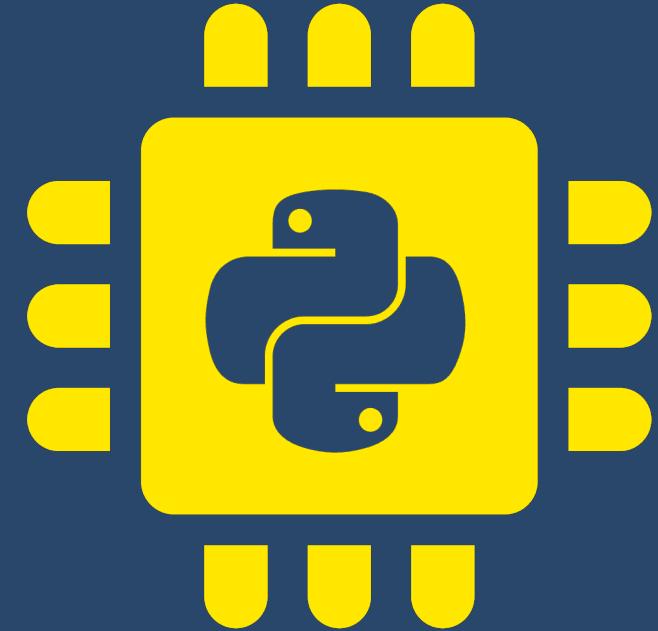


Вновь ускоряем сри-bound задачи



Денис Аникин
<https://xfenix.ru>

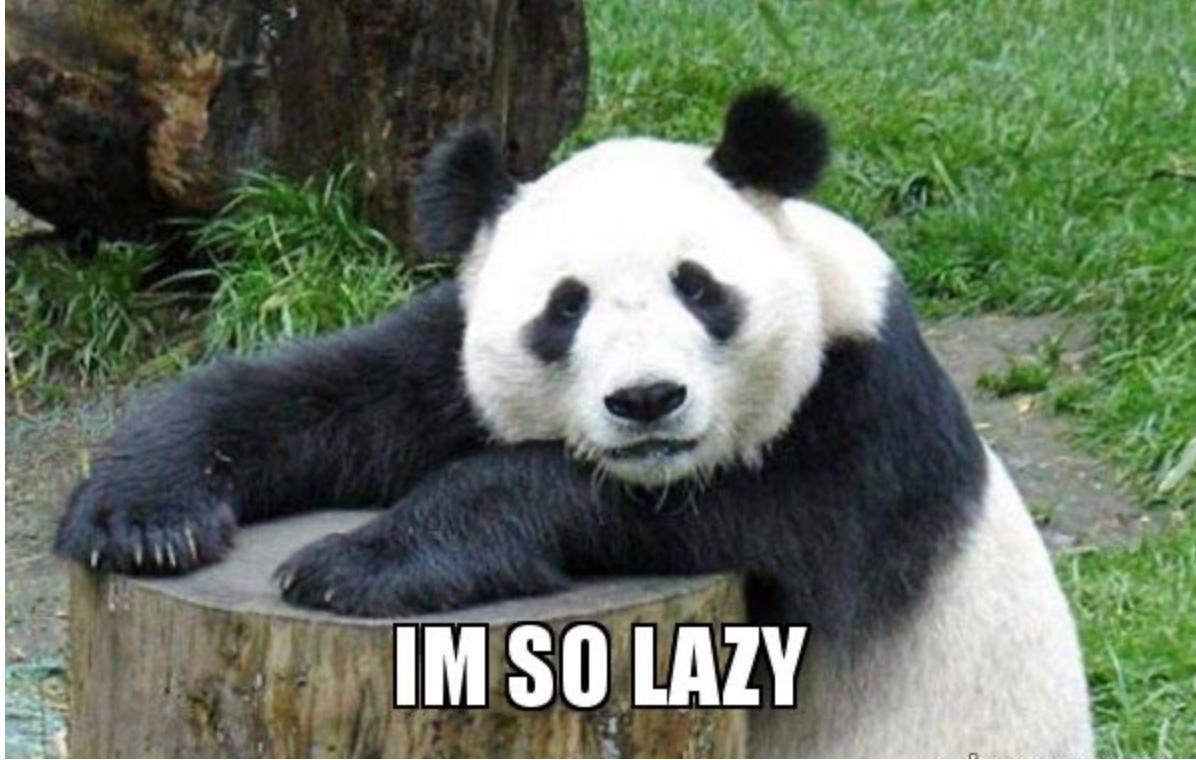
Почему «вновь»?

Я не буду говорить о
специализированных
решениях вроде пнтр

Приготовьтесь!
Сегодня будет **15** способов
ускорить ваш сри-bound код

**Мы разобъем их на 4
категории**





IM SO LAZY

Стандартная библиотека



Что в комплекте?

Тут все довольно банально

- multiprocessing!
- concurrent.futures



Пример!

```
from multiprocessing import Pool

INPUT_VALUES = (1, 100, 400, 55, 33, 44, 11, 15, 16, 17)
PROC_COUNT = 3
CHUNK_PARTS = len(INPUT_VALUES) // PROC_COUNT

def _chunks(l, n):
    for i in range(0, n):
        yield l[i::n]

def run_heavy_task(parts):
    print(parts)
    return [item * item for item in parts]

def run_example():
    answers = []
    ready_chunks = _chunks(INPUT_VALUES, CHUNK_PARTS)
    with Pool(PROC_COUNT) as pool:
        answers.extend(pool.map(run_heavy_task, ready_chunks))
    print(answers)

if __name__ == '__main__':
    run_example()
```

Плюсы и минусы

- ✓ В комплекте!
- ✓ Очень очень много всего разного!
Например, multiprocessing.connection

Плюсы и минусы

- ✓ В комплекте!
- ✓ Очень очень много всего разного!
Например, `multiprocessing.connection`

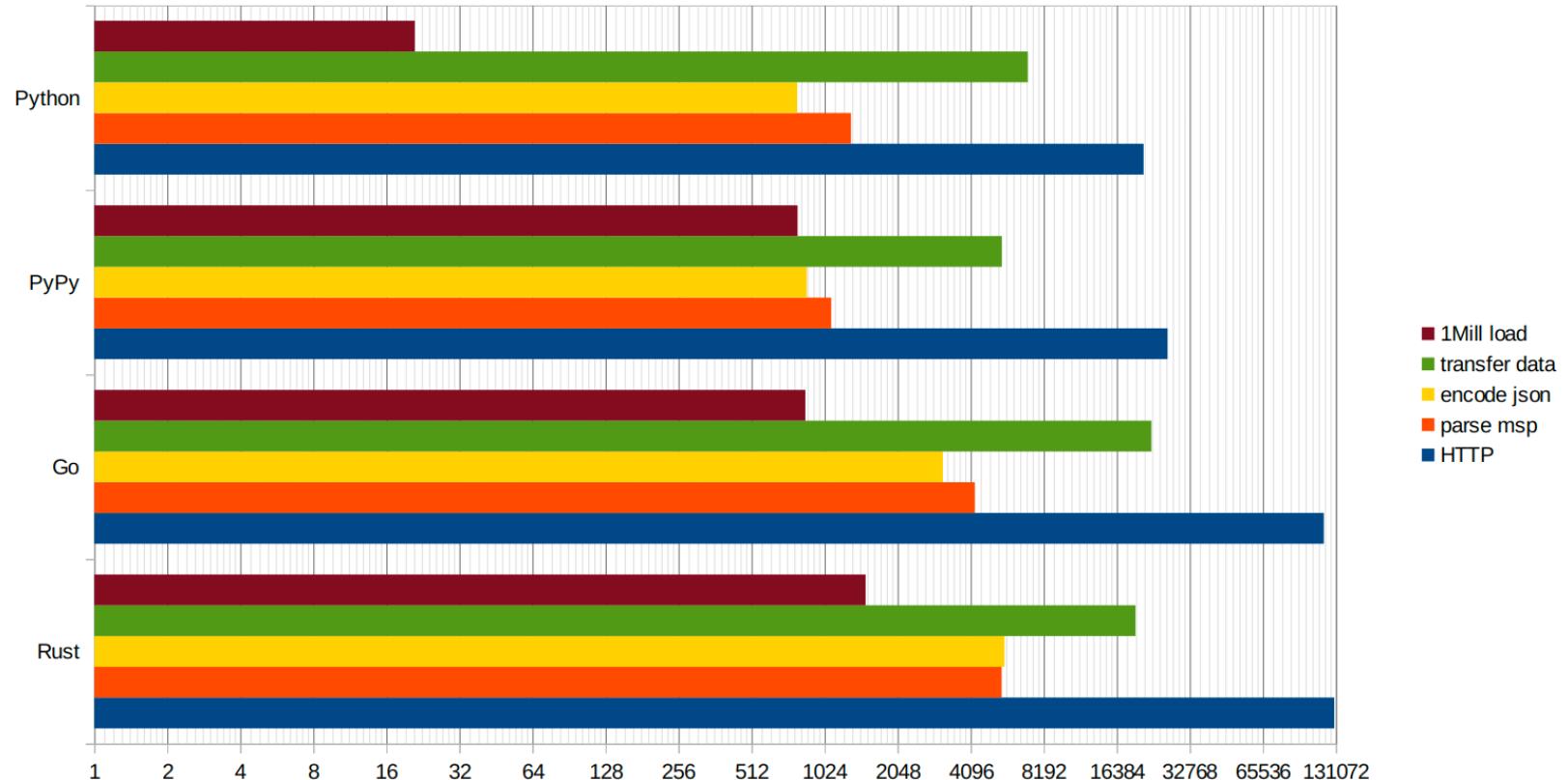
- ✗ Надо реализовывать вручную
- ✗ Inter Process Communication
- ✗ Все минусы конкретного кода
- ✗ Непросто
- ✗ Тяжело поддерживать

**Категория 1:
другие интерпретаторы**

x

PyPy

<https://deavid.wordpress.com/2019/10/12/benchmarking-python-vs-pypy-vs-go-vs-rust/>

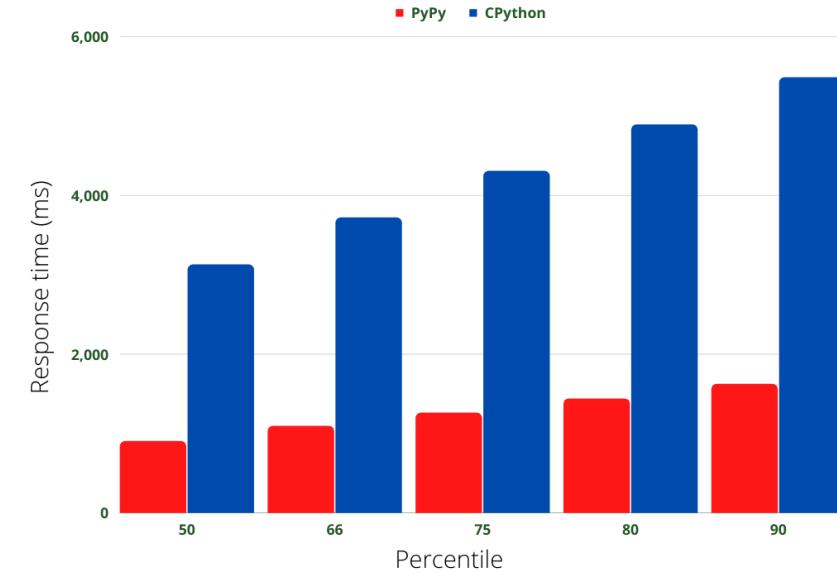
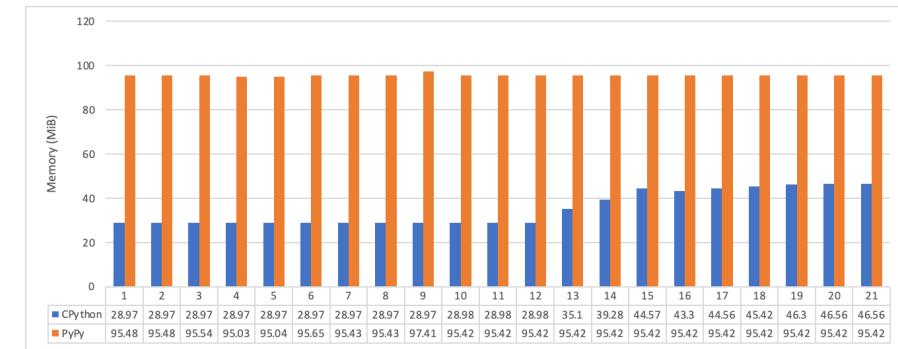
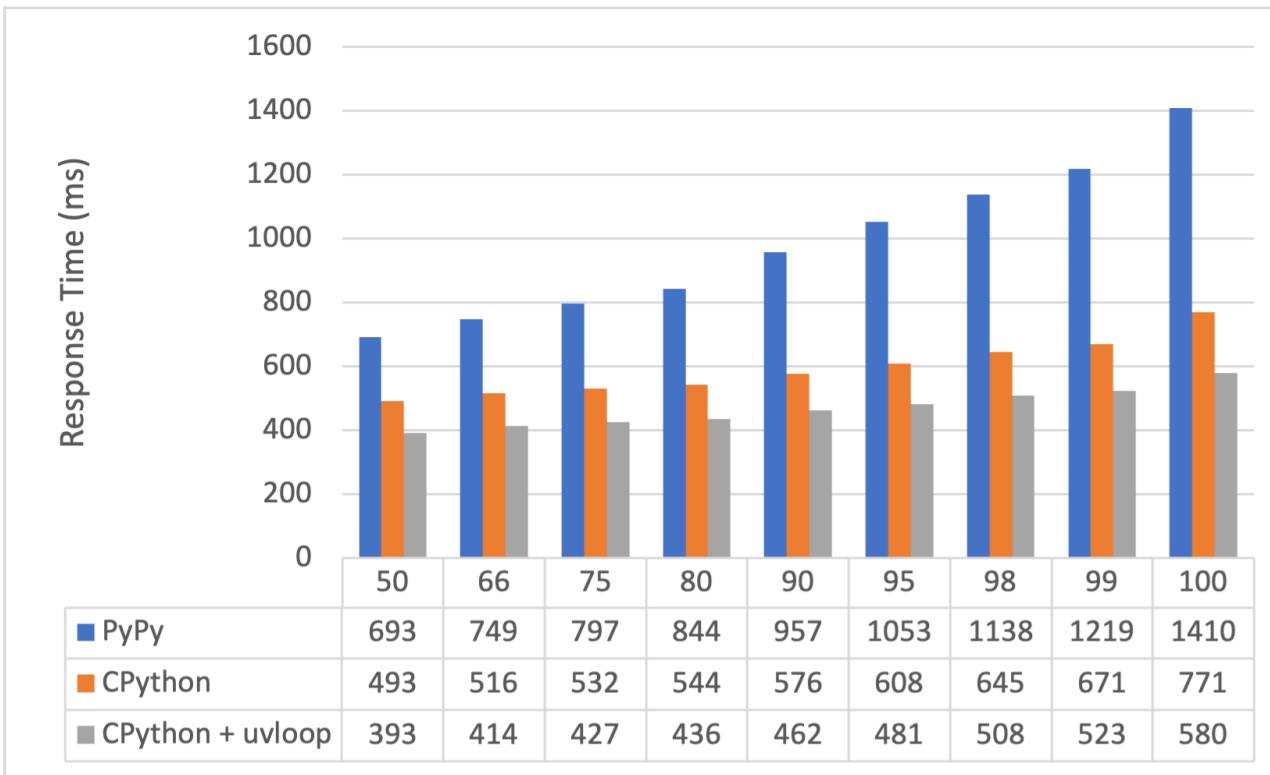




<https://codesolid.com/pypy-first-look-a-faster-version-of-python/>

Benchmark	Time in CPython	Time in PyPy	Difference
10000 x 10000 numbers in list iterated	0.82558875	0.236211458	3.50 times faster
10000 x 10000 numbers in array iterated	1.369477958	0.228980292	5.98 times faster
Python sum on a list of 256,000 numbers	0.597	0.29	2.06 times faster
Python sum on a bytearray of 256,000 numbers	0.544	0.633	1.16 times slower
Python sum on numpy array 256,000 numbers	4.8544	74.7397	15.40 times slower
Numpy sum on numpy array 256,000 numbers	0.05	0.056	1.12 times slower

<https://tonybaloney.github.io/posts/pypy-in-production.html>



Плюсы и минусы

 Очень просто!*

Плюсы и минусы

✓ Очень просто!*

*не всегда

- ✗ Потребляет много памяти
- ✗ Далеко не во всех сценариях есть разница в плюс
- ✗ Специализация на математике (?)
- ✗ Сторонний интерпретатор
- ✗ Отстает от Cpython
- ✗ ABI несовместим ==
перекомпиляция расширений (full pyston)

Pyston



We've heard many people ask for better Python performance, but our experience seems to say that a **full alternative implementation** is not a particularly appealing solution to this ask

	pyperformance	macrobenchmarks
Pyston (full) 2.3.5	+65%	+28%
pyston-lite 2.3.5 on 3.8	+28%	+10%
CPython 3.11rc2	+26%	+12%

**Иными словами pyton
2.3.5 vs python 3.11 ~
14% разницы**



Пример!

```
pip install pyston_lite_autoload
```

Плюсы и минусы

- ✓ Основная ветка — вроде бы просто
- ✓ Lite — супер просто!

Плюсы и минусы

- ✓ Основная ветка — вроде бы просто
- ✓ Lite — супер просто!

- ✗ Опять сторонний интерпретатор
- ✗ Форк 3.8
- ✗ Не очень большой буст
- ✗ pyston lite дает еще меньше
- ✗ Документации нет, примеров нет

А стало ли быстрее?

```
→ ekbpy git:(main) pip install pyston_lite_autoload
```

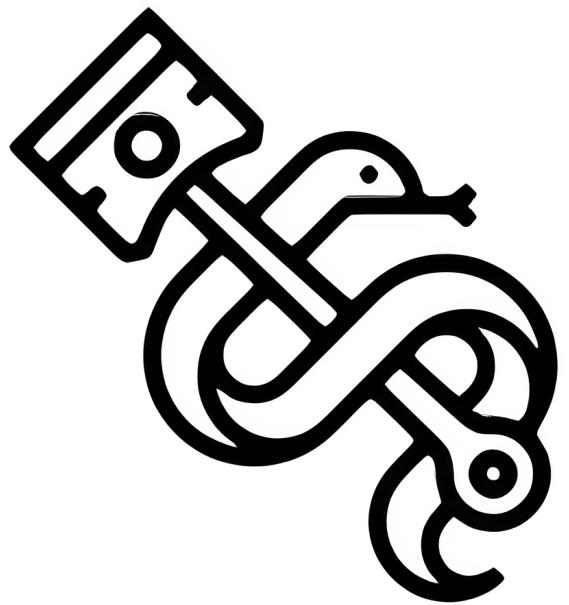
```
→ ekbpy git:(main) x python raw.py
```

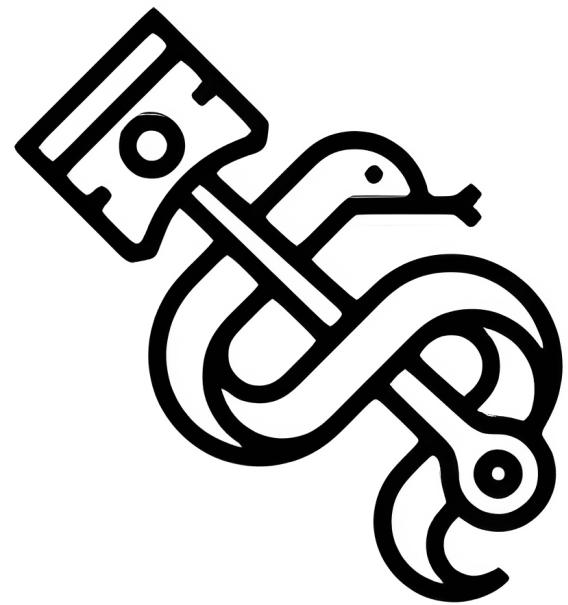
```
2.532589207869023
```

```
→ ekbpy git:(main) x pip uninstall pyston_lite_autoload
```

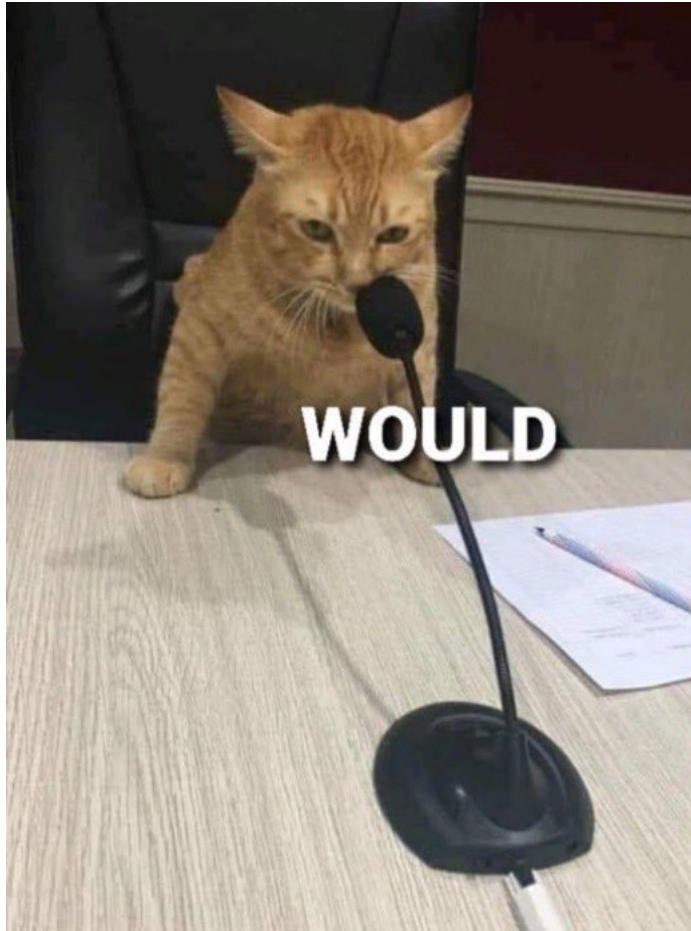
```
→ ekbpy git:(main) x python raw.py
```

```
4.179197291843593
```





О, работает!



Подводим итоги 1
категории
(интерпретаторы)

х

Мой выбор —
оба хороши



Категория 2: компиляторы

MyPyC

Пример!

- pip install **mypy**
- mypyc **my_fancy.py**

Плюсы и минусы

- ✓ Просто!
- ✓ Быстро

Плюсы и минусы

✓ Просто!

✓ Быстро

✗ Не помогает 😞

Плюсы и минусы

✓ Просто!

✓ Быстро

✗ Не помогает 😞

*не всегда помогает

Плюсы и минусы

- ✓ Просто!
- ✓ Быстро
- ✓ Куча плюсов относительно cython

✗ Не помогает 😞
*не всегда помогает

```
tar: warning: directory not found for option -C /usr/local/opt/openssl/lib
copying build/lib.macosx-12.4-arm64-3.9/ekbpy/raw_mupyc.cpython-39-darwin.so → ekbpy
error: could not create 'ekbpy/raw_mupyc.cpython-39-darwin.so': No such file or directory
→ ekbpy git:(main) ✘ pip install -U mupyc
```

Pycom

Ссылка

Опять без QR

<https://github.com/Omyyyy/pycom>

Плюсы и минусы

 Относительно просто

Плюсы и минусы

✓ Относительно просто

- ✗ Эффект не ясен
- ✗ Далеко все можно компилировать
- ✗ В pip нет
- ✗ У меня просто не работает. Тикет про range.hpp полгода уже лежит без движения

```
→ ekbpy git:(main) ✘ pycom raw.py
error: module/library _base is not implemented yet, sorry'
→ ekbpy git:(main) ✘ pycom _base.
[ERROR]: '_base.' not found
→ ekbpy git:(main) ✘ pycom _base.py
error: module/library timeit is not implemented yet, sorry'
→ ekbpy git:(main) ✘ pycom _base.py
pycom: CompilationError:
          g++: 'headers/other/range.hpp' file not found
          g++: #include "headers/other/range.hpp"
          g++: ^~~~~~
          g++: 1 error generated.
```

А эффект?

Benchmark	C <code>Python</code>	Pycom	Pycom with --fastmath	pypy
Multiples of 3 and 5	9.383s	0.133s	0.106s	0.495s
Primes	17.127s	4.441s	3.994s	4.577s
Stack Operations	8.857s	2.132s	1.992s	3.113s

x

Codon

Полезные ссылки

<https://github.com/exaloop/codon>

« Typical speedups over Python are on the order of **10-100x** or more, on a **single thread**. Codon's performance is typically on par with (and sometimes better than) that of C/C++. Unlike Python, Codon supports **native multithreading**, which can lead to speedups many times higher still

« Codon is a Python-compatible language, and many Python programs will work with few if any modifications



Плюсы и минусы

✓ Очень много возможностей

✗ Другой как бы язык
✗ Опять не для всех кейсов
✗ Есть отдельный пакет-джитилка
✗ Не работающий инсталлятор под
линукс

А стало ли быстрее?

Да вроде?

```
→ ekbpy git:(main) ✘ python codon_.py  
4.026413624997076  
→ ekbpy git:(main) ✘ python codon_.py  
3.982390416000271  
→ ekbpy git:(main) ✘ python raw.py  
4.194220249999489
```



x

Nuitka

Нютка

Нуитка



Пример!

- pip install **nuitka**
- python -m **nuitka** my_fancy_file.py

Полезные ссылки

- Nuitka the python compiler: <https://www.youtube.com/watch?v=ZDHkla5rlIg>
- Nuitka speedcenter: <https://speedcenter.nuitka.net/>

Плюсы и минусы

- ✓ Просто работает!
- ✓ Очень легко
- ✓ Есть в pip

✗ Можно подорваться на ARM/mac m1

А стало ли быстрее?

Нуу...да?

```
→ ekbpy git:(main) ✘ python nuitka_.py
4.416637832997367
→ ekbpy git:(main) ✘ python raw.py
4.504299208987504
    . . . . .
```



Подводим итоги 2
категории
(компиляторы)

х

Мой выбор —
nuitka

Категория 3: диалекты

Cython



**Use Cython to get more than 30X speedup on
your Python code**

Что в комплекте?

Тут все довольно банально

- pip install cython
- cythonize my_file.py

```
def f(x: cython.double):
    return x ** 2 - x

def integrate_f(a: cython.double, b: cython.double, N: cython.int):
    i: cython.int
    s: cython.double
    dx: cython.double
    s = 0
    dx = (b - a) / N
    for i in range(N):
        s += f(a + i * dx)
    return s * dx
```

```
def f(double x):
    return x ** 2 - x

def integrate_f(double a, double b, int N):
    cdef int i
    cdef double s
    cdef double dx
    s = 0
    dx = (b - a) / N
    for i in range(N):
        s += f(a + i * dx)
    return s * dx
```

Плюсы и минусы

- ✓ Контролируемо
- ✓ Хорошая скорость (иногда, видимо, выдающаяся)
- ✓ Хорошая документация

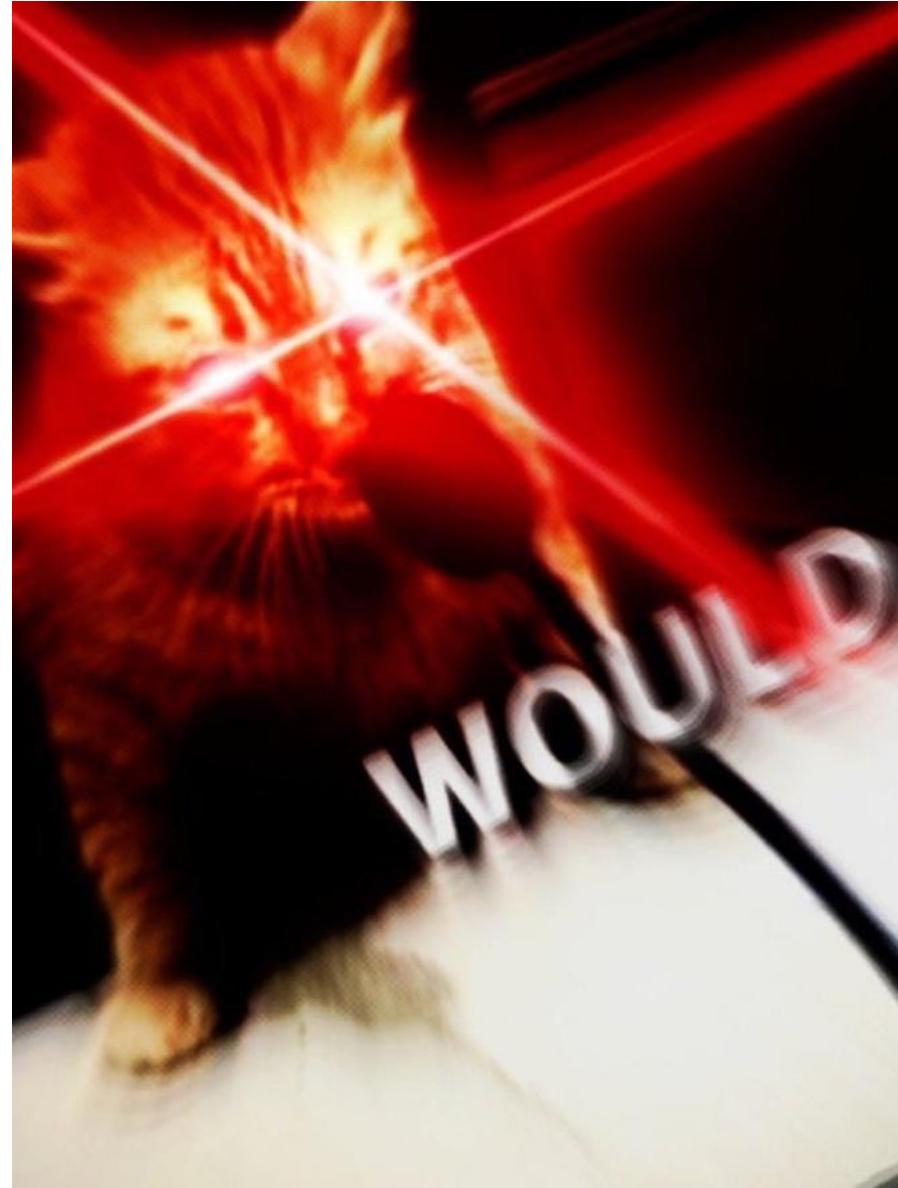
Плюсы и минусы

- ✓ Контролируемо
- ✓ Хорошая скорость (иногда, видимо, выдающаяся)
- ✓ Хорошая документация
- ✗ Учить немного другой язык (диалект)
- ✗ Зависимости при сборке
- ✗ Более сложная сборка пакетов
- ✗ Если не знаете C, то сложно использовать
- ✗ Документации недостаточно, чтобы программировать

А стало ли быстрее?

Очень даже! 111

```
→ ekpy git:(main) ✘ python cython_.py  
1.9537809169851243  
→ ekpy git:(main) ✘ python raw.py  
4.170039375079796
```



PeachPy

Пример!

```
from peachpy import *
from peachpy.x86_64 import *

x = Argument(int32_t)
y = Argument(int32_t)

with Function("Add", (x, y), int32_t) as asm_function:
    reg_x = GeneralPurposeRegister32()
    reg_y = GeneralPurposeRegister32()

    LOAD.ARGUMENT(reg_x, x)
    LOAD.ARGUMENT(reg_y, y)

    ADD(reg_x, reg_y)

    RETURN(reg_x)

python_function = asm_function.finalize(abi.detect()).encode().load()

print(python_function(2, 2)) # -> prints "4"
```

ASSEMBLER



x86-64 assembler embedded in Python

Плюсы и минусы

- УэээУээ Прикольно?
- Возможно очень быстро

Плюсы и минусы

- УэээУээ Прикольно?
- Возможно очень быстро

 IQ 450

Typed python

Полезные ссылки

- https://github.com/APrioriInvestments/typed_python

Пример!

```
import _base
from typed_python import Entrypoint, ListOf

@Entrypoint
def run_example():
    output_buf = ListOf(int]()
    for one_number in range(100):
        output_buf.append(one_number * one_number)
    return output_buf

if __name__ == "__main__":
    _base.run_timeit()
```



Плюсы и минусы

- ✓ Это то, чего я так долго ждал
- ✓ Статическая типизация в рантайме!
- ✓ Классные структуры данных!

Плюсы и минусы

- ✓ Это то, чего я так долго ждал
- ✓ Статическая типизация в рантайме!
- ✓ Классные структуры данных!

- ✗ Не работает
- ✗ Нет документации
- ✗ Слабая поддержка комьюнити

```

asm("int3");

<inl>::<lambda>::<lambda>(auto&316>; PyObject *object) const here
    typed_python/PyInstance.hpp:316:30:   required from 'static PyObject*
onst char*, const char*>::<lambda(auto:316>; PyObject = _objec
    typed_python/PyInstance.cpp:433:14:   required from here
        typed_python/PyRegisterTypeInstance.hpp:365:45: warning: '*' in bool [-ferror-limit
        | 365 |     return registerValueToPyObject(T(self*other));
        |         |~~~~~^~~~~~|de 1
        | /tmp/ccxGsXAS.s: Assembler messages:
        | /tmp/ccxGsXAS.s:174606: Error: unknown mnemonic 'int3' -- 'int3'
        | /tmp/ccxGsXAS.s:189052: Error: unknown mnemonic 'int3' -- 'int3'
        | /tmp/ccxGsXAS.s:189716: Error: unknown mnemonic 'int3' -- 'int3'
        | /tmp/ccxGsXAS.s:198947: Error: unknown mnemonic 'int3' -- 'int3'
        | /tmp/ccxGsXAS.s:232264: Error: unknown mnemonic 'int3' -- 'int3'
        | /tmp/ccxGsXAS.s:232560: Error: unknown mnemonic 'int3' -- 'int3'
        | /tmp/ccxGsXAS.s:958083: Error: unknown mnemonic 'int3' -- 'int3'
        | /tmp/ccxGsXAS.s:2068217: Error: unknown mnemonic 'int3' -- 'int3'
        | /tmp/ccxGsXAS.s:2238470: Error: unknown mnemonic 'int3' -- 'int3'
        error: command '/usr/bin/gcc' failed with exit code 1
[end of output]

note: This error originates from a subprocess, and is likely not a problem
ERROR: Failed building wheel for typed-python
Building wheel for psutil (pyproject.toml): started
Building wheel for psutil (pyproject.toml): finished with status 'done'
Created wheel for psutil: filename=psutil-5.9.4-cp310-abi3-linux_aarch64.
Stored in directory: /root/.cache/pip/wheels/54/8ff/fb/a70448250cac755f3a/
Successfully built psutil
Failed to build typed-python

```

Numba

Пример!

```
from numba import njit
import random

@njit
def monte_carlo_pi(nsamples):
    acc = 0
    for i in range(nsamples):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            acc += 1
    return 4.0 * acc / nsamples
```

Плюсы и минусы

 Ускоряет?

Плюсы и минусы

- ✓ Ускоряет?
- ✓ JIT!?

- ✗ Другой как бы язык
- ✗ Не для каждого кейса

А стало ли быстрее?

Вполне! И это мои так себе примеры без использования диалекта...

```
→ ekbpy git:(main) ✘ python numba_.py
1.5403167500626296
→ ekbpy git:(main) ✘ python raw.py
4.223473167046905
```

x

Taichi



Пример

```
import taichi as ti
import taichi.math as tm

ti.init(arch=ti.gpu)

n = 320
pixels = ti.field(dtype=float, shape=(n * 2, n))

@ti.func
def complex_sqr(z): # complex square of a 2D vector
    return tm.vec2(z[0] * z[0] - z[1] * z[1], 2 * z[0] * z[1])

@ti.kernel
def paint(t: float):
    for i, j in pixels: # Parallelized over all pixels
        c = tm.vec2(-0.8, tm.cos(t) * 0.2)
        z = tm.vec2(i / n - 1, j / n - 0.5) * 2
        iterations = 0
        while z.norm() < 20 and iterations < 50:
            z = complex_sqr(z) + c
            iterations += 1
        pixels[i, j] = 1 - iterations * 0.02

gui = ti.GUI("Julia Set", res=(n * 2, n))

for i in range(1000000):
    paint(i * 0.03)
    gui.set_image(pixels)
    gui.show()
```

Плюсы и минусы

- ✓ Вроде бы работает
- ✓ Быстро?
- ✓ Много вариантов (гри, сри и т.п.)

Плюсы и минусы

- ✓ Вроде бы работает
- ✓ Быстро?
- ✓ Много вариантов (гри, сри и т.п.)

- ✗ Опять arm?
- ✗ Могли бы написать про зависимость от zstd
- ✗ Не для каждого сри bound кейса
- ✗ Другой как бы язык

А стало ли быстрее?

Нуу...

```
→ ekbpy git:(main) ✘ python taichi_.py
[Taichi] version 1.4.0, llvm 16.0.0git,
[Taichi] Starting on arch=arm64
/Users/xfenix/web/ekbpy/taichi_.py:18: U
y not work as expected. Proceed with cau
    output_buf.append(one_number * one_num
5.116075500147417
```

```
→ ekbpy git:(main) ✘ python taichi_.py
[Taichi] version 1.4.0, llvm 16.0.0git, co
[Taichi] Starting on arch=arm64
17.658035999862477
```

Подводим итоги 3 категории (диалекты)

Мой выбор —
cython + taichi + numba



х

Категория 4: другие языки (совсем)

x

C



Полезные ссылки

- <https://docs.python.org/3/extending/extending.html>

if you know how to program in C.

Плюсы и минусы

- ✓ А?
- ✓ Быстро!

Плюсы и минусы

✓ А?

✓ Быстро!

✗ С

✗ Сборка ☹

✗ Сборка ☹ ☹

Подводим итоги 3 категории (диалекты)

Мой выбор — taichi и
cython!

x

Gopy

Пример!

```
$ go mod init dummy.com/dum
$ go get github.com/go-python/gopy/_examples/hi
$ gopy build -output=out -vm=python3 github.com/go-python/gopy/_examples/hi
```

Плюсы и минусы

 Удобно и быстро

 Рантайм го

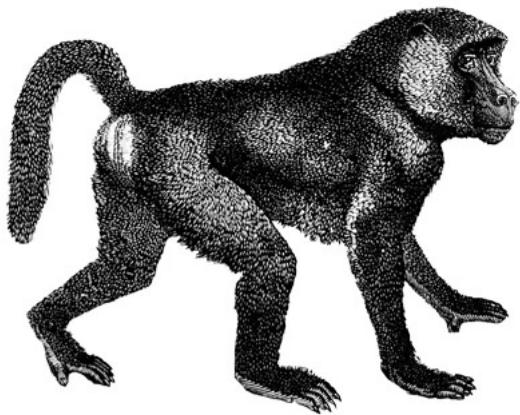
 Другой язык

 Сборка!

x

Maturin

Time for change.



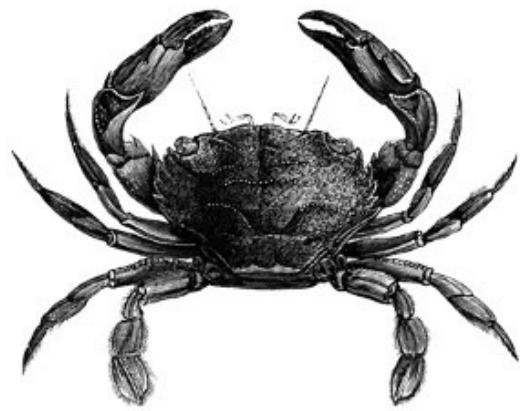
Rewrite everything in Rust

10x Developer Guide

O RLY?

by Bootcamp Graduate

Only one mutable borrow at a time

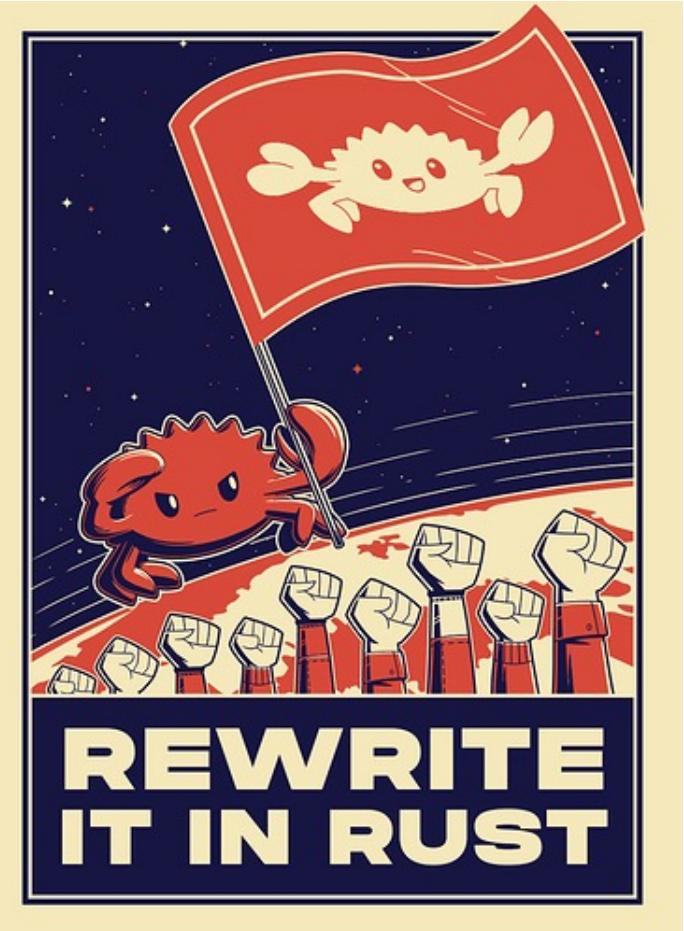


Rewriting it in Rust

Banning unsafe code

O RLY?

RESF



Пример!

```
pip install maturin
```

Mixed rust/python projects

To create a mixed rust/python project, create a folder with your module name (i.e. `lib.name` in `Cargo.toml`) next to your `Cargo.toml` and add your python sources there:

```
my-project
├── Cargo.toml
├── my_project
│   ├── __init__.py
│   └── bar.py
├── pyproject.toml
└── README.md
└── src
    └── lib.rs
```

You can specify a different python source directory in `pyproject.toml` by setting `tool.maturin.python-source`, for example

`pyproject.toml`

```
[tool.maturin]
python-source = "python"
```

then the project structure would look like this:

```
my-project
├── Cargo.toml
├── python
│   └── my_project
│       ├── __init__.py
│       └── bar.py
├── pyproject.toml
└── README.md
└── src
    └── lib.rs
```

Плюсы и минусы

- Это очень быстро
- Это модно

Плюсы и минусы

- ✓ Это очень быстро
- ✓ Это модно

- ✗ Сложный язык
- ✗ Много мета-информации
- ✗ Сборка!1!1

А стало ли быстрее?

Нуу...

Прогоняем бенчмарки

- На тестовом примере оказалось, что python быстрее rust

Подводим итоги 4
категории
(другие языки)

Мой выбор — gory

Сводим вместе!



Я этого любителя
в экселе видел!
Плохие таблицы
составляет!



Выбираем финалистов

Решение	Из кого выбирал	Финалист
Интерпретаторы	PyPy, Pyston	PyPy + Pyston lite
Компиляторы	MypyC, PyCom, Codon, Nuitka	Nuitka
Диалекты	Cython, Numba, Taichi, PeachPy, Typed python	Cython + Taichi + Numba
Другие языки	C, Gopy, Maturin (Rust)	Gopy

Послесловие

Python 3.14 Will be Faster than C++

Benchmarking the new and impressive Python 3.11



Of course

Что-то и ОЧЕНЬМНОГО-кратное ускорение Python-кода

Блог важной компании



Of course

Совсем уже напоследок

Немного performance специализированных библиотек (я хотел сделать вид, что знаю не только NumPy)

- Ruff
- Polars
- Robyn

Адрес репы, где есть презентация

<https://github.com/xfenix/ekbpy2023>





Спасибо



<https://xfenix.ru>

<https://github.com/xfenix>

<https://www.linkedin.com/in/xfenix/>