



NEW
PRO
LAB

Python для Data Engineer'a

Николай Марков
@enchantner

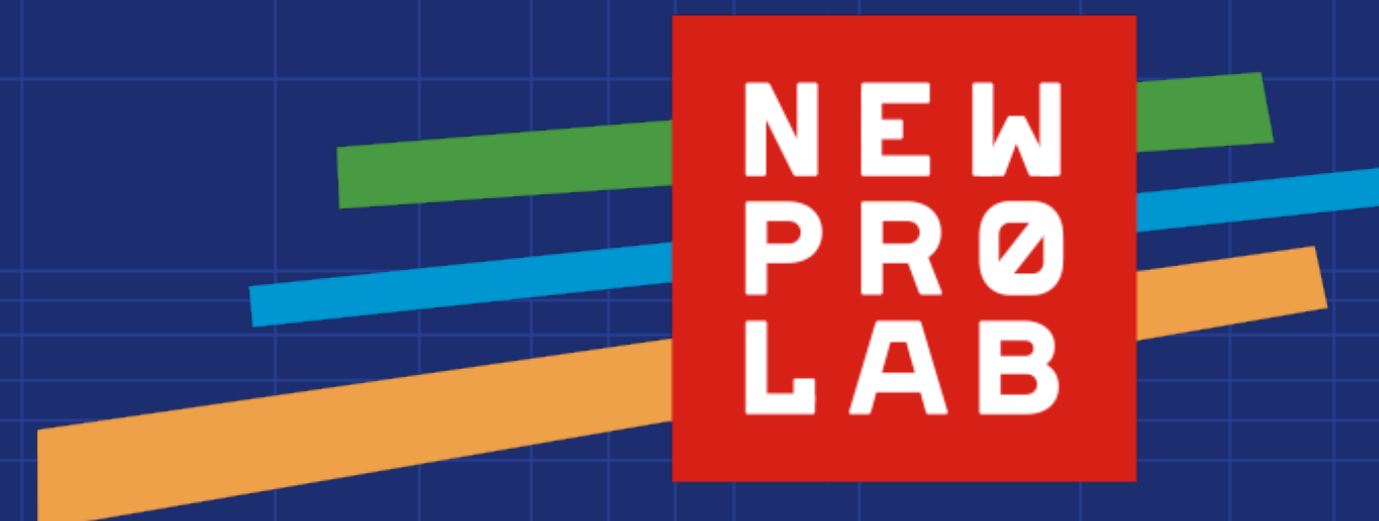


NEWPROLAB.COM

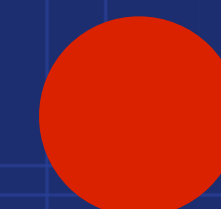
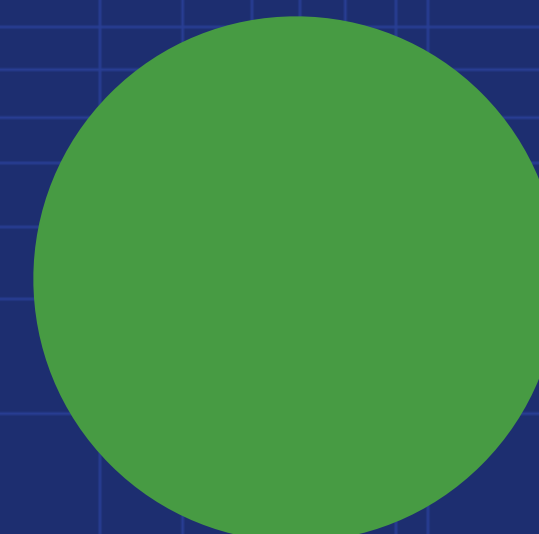
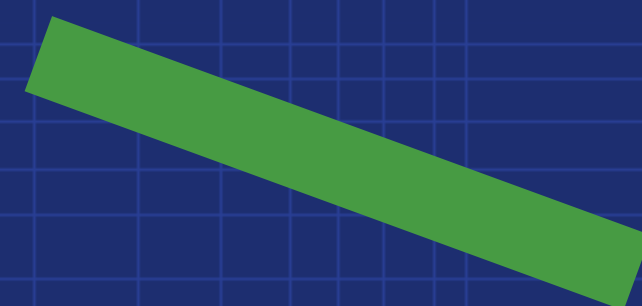


Типичные задачи

- Сбор и очистка данных
- ETL и конвертация форматов
- Параллелизация и ускорение кода
- Создание аналитических архитектур и выстраивание пайплайнов
- Все то, что не связано непосредственно с отладкой моделей и математикой



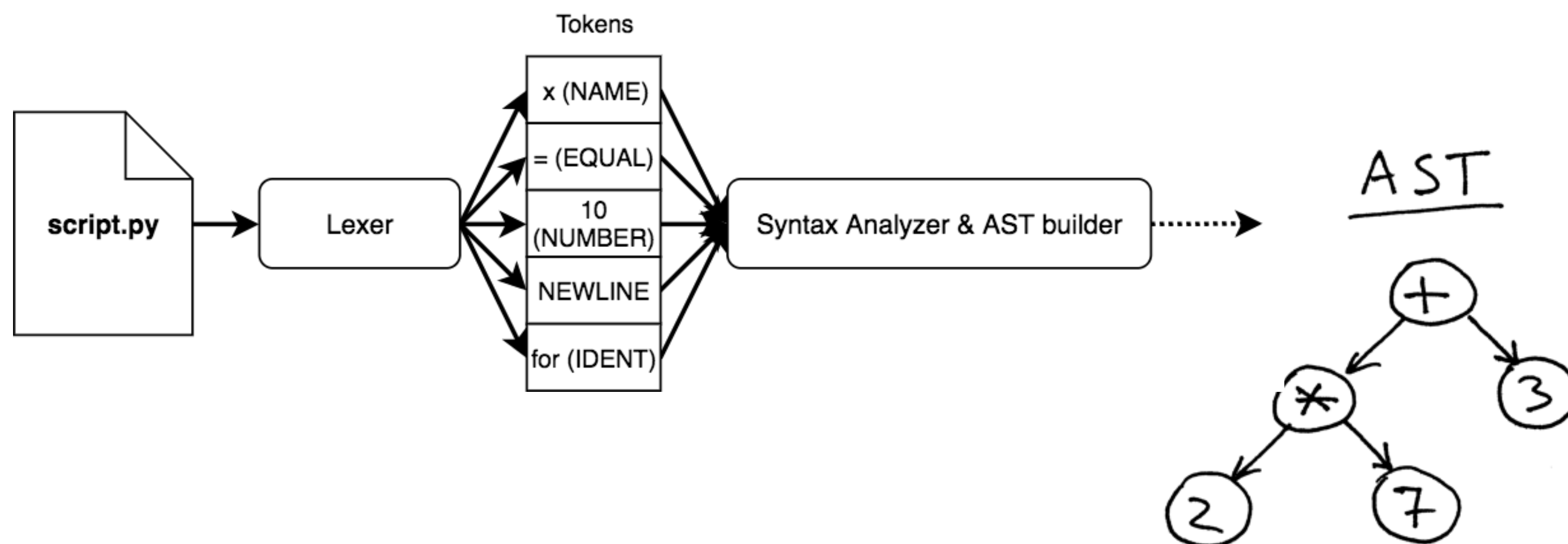
Параллельность, GIL



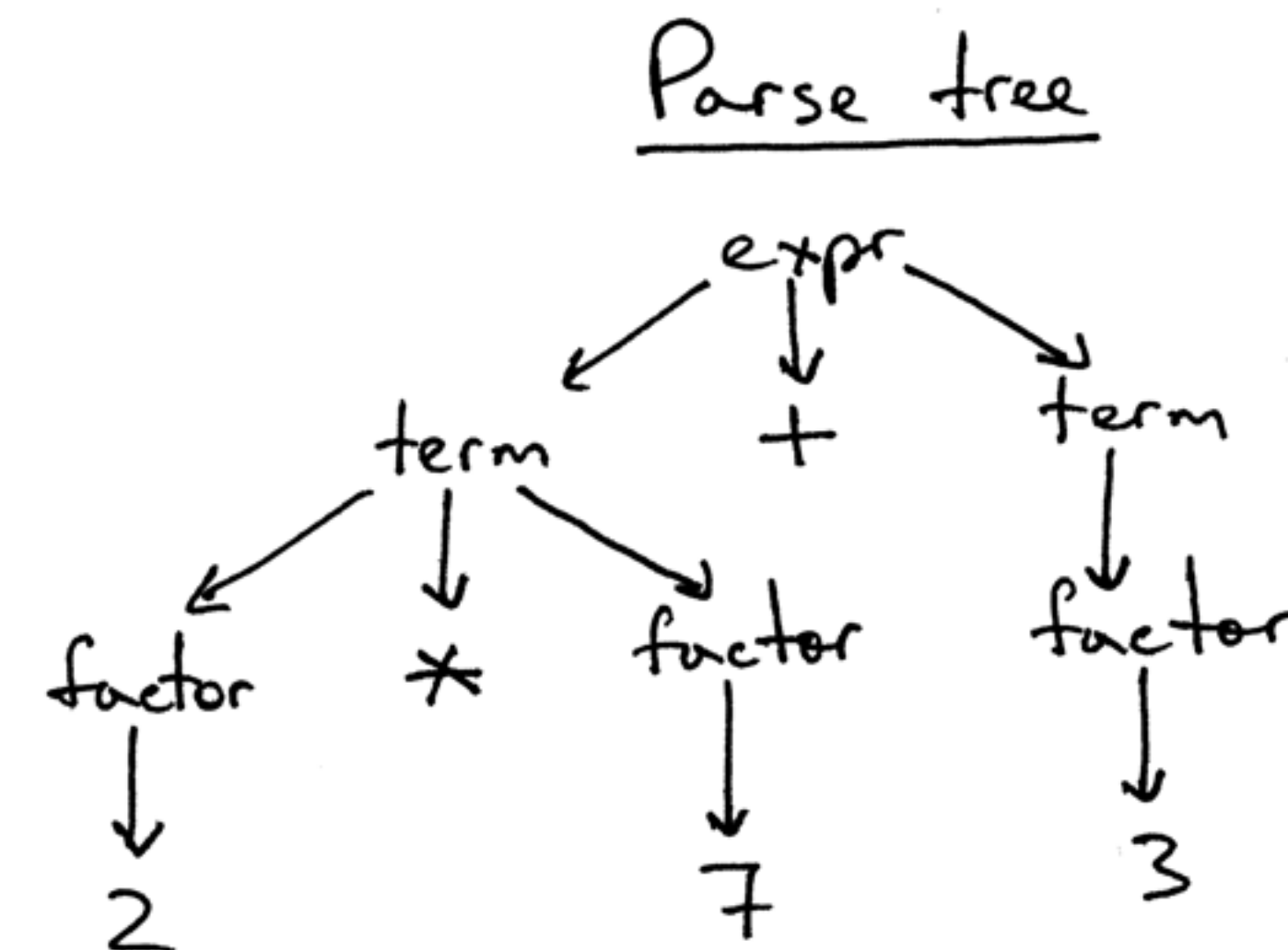
Что такое потоки?

- Системный планировщик отдает процессорное время потокам/процессам, переключая между ними контекст
- Процессы/потоки работают "параллельно", в идеале используя несколько ядер процессора
- Пути планировщика неисповедимы, нельзя заранее предсказать, какой процесс получит ресурсы в конкретный момент
- Потоки надо синхронизировать согласно задачам, чтобы не было проблем с одновременным доступом
- Пример - простая версия веб-сервера
- Есть CPU-bound задачи и есть I/O-bound задачи - важно понимать разницу

Чуть больше про кишки



`2 * 7 + 3`



0	LOAD_CONST	1	(2)
3	LOAD_CONST	1	(2)
6	BINARY_MULTIPLY		
7	PRINT_ITEM		
8	PRINT_NEWLINE		
9	LOAD_CONST	0	(None)
12	RETURN_VALUE		

Что такое GIL?

- GIL - это глобальный мьютекс (механизм синхронизации) в интерпретаторе Python
- GIL запрещает выполнять байткод Python больше чем одному потоку одновременно
- Но это касается ТОЛЬКО байткода Python и не распространяется на I/O операции
- Потоки Python (в отличие от потоков, скажем, в Ruby) - это полноценные потоки ОС

<https://pastebin.com/iwA5uyVV>

<https://pastebin.com/VEhbv3jQ>



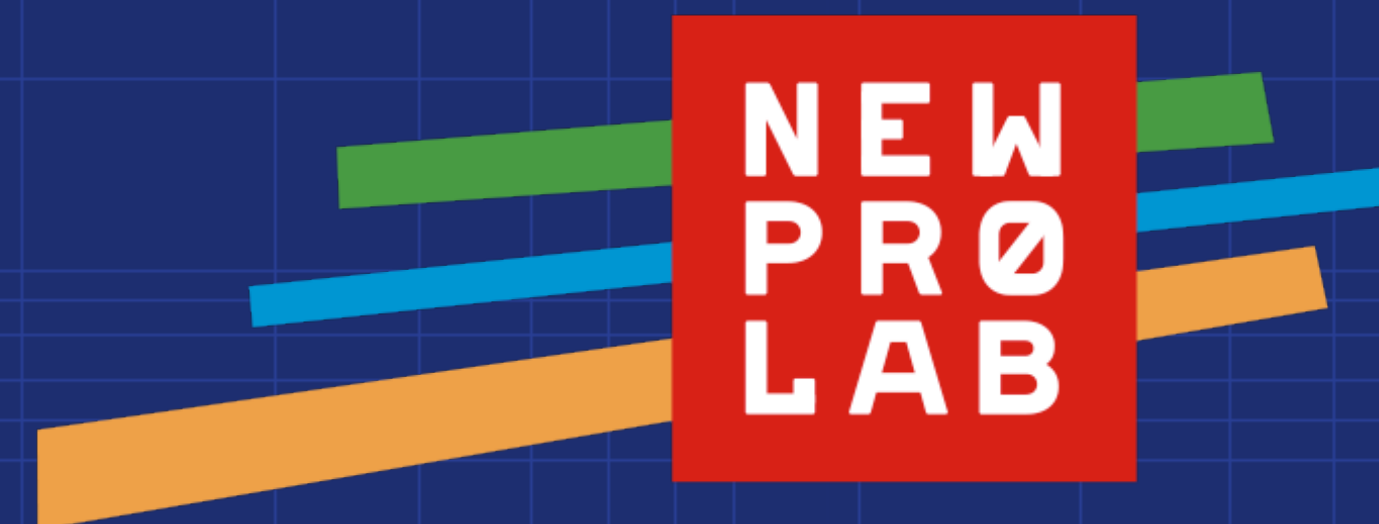
Упражнение - sleepsort

Предположим у нас есть **короткий** список чисел от 0 до 10. Чтобы их вывести в отсортированном порядке - достаточно каждый поток заставить "спать" количество секунд, равное самому числу, и только потом его выводить. В чем недостаток данного подхода?

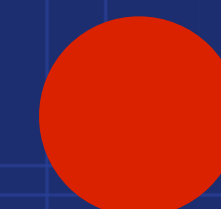
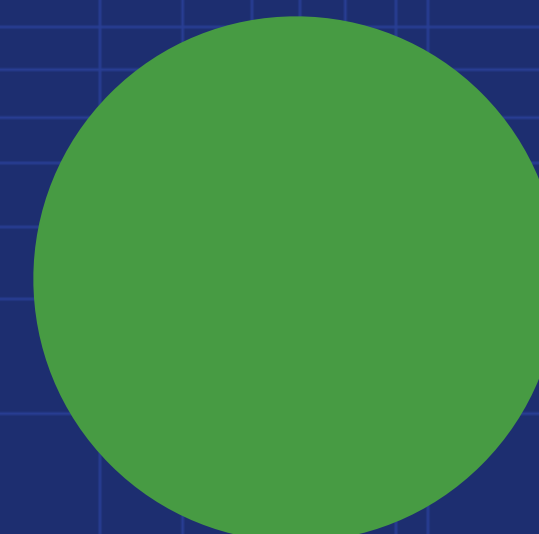
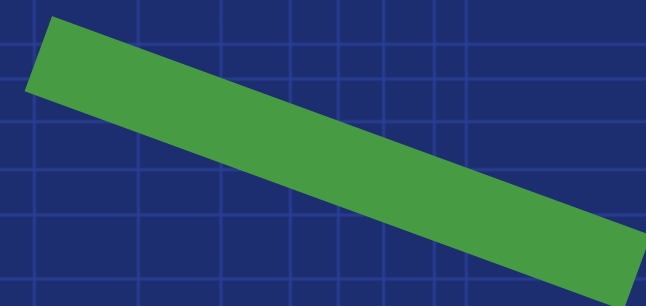
Процессы вместо потоков?

```
from multiprocessing import Process  
from multiprocessing import Queue  
concurrent.futures.ThreadPoolExecutor
```

- Большие затраты на пересылку и синхронизацию



Асинхронность



Что такое асинхронность?

- Параллельность - это выполнение двух фрагментов кода одновременно.
- Асинхронность - это выполнение кода НЕ последовательно.
- Асинхронность может быть реализована с помощью параллельности, а может - с помощью ручного переключения контекста в самом коде, с сохранением последнего состояния. Кто сказал `yield`?
- Когда куски кода сами решают, когда передавать управление друг другу, и не зависят от внешнего системного планировщика, то это называется "кооперативной многозадачностью", а эти куски кода - корутинами или сопрограммами.
- Недостаток - долгоиграющая процедура НЕ под контролем `event loop`'а вешает вообще ВСЕ

Событийно-ориентированное программирование

- Две основные составляющие асинхронного кода - это event loop (цикл отлова событий) и корутины
- Пока корутина ждет внешнее событие - контекст переключается на другую
- Помимо переключения контекста корутины могут отправлять друг другу сообщения
- К сожалению, в современной реализации асинхронности в Python обычные и асинхронные функции не являются взаимозаменяемыми
- Альтернативные реализации для старых версий - Gevent, Eventlet и Tornado. И еще несколько.

Классический подход - генераторы

```
import random
```

```
def multiply_gen(lst):  
    multiplier = 2  
    for i in lst:  
        multiplier = (yield multiplier) or multiplier  
        print(f"Multiplier is now {multiplier}")
```

```
data = [random.randint(-10, 10) for _ in range(10)]  
mult_data = multiply_gen(data)  
for i, entry in enumerate(zip(data, mult_data)):  
    print(f"{i}: {entry} {entry[0] * entry[1]}")  
    if i == 4:  
        mult_data.send(3)  
        print(f"{i}: {entry} {entry[0] * entry[1]}")
```



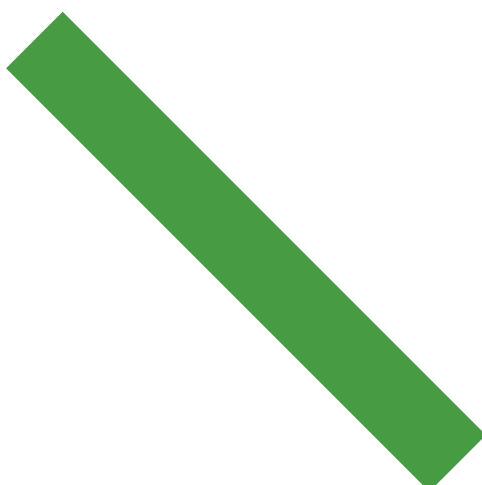
NEW
PRO
LAB

Классический подход - генераторы



<http://www.dabeaz.com/finalgenerator/>

<http://www.dabeaz.com/coroutines/index.html>



Asyncio

```
import asyncio
```

```
asyncio.Queue() # асинхронная очередь
```

```
asyncio.sleep(10) # асинхронный "сон"
```

```
# асинхронный subprocess
```

```
asyncio.create_subprocess_exec()
```

```
asyncio.Lock() # асинхронный мьютекс
```

```
# ручное добавление корутины в event loop
```

```
asyncio.ensure_future()
```

```
# дождаться окончания работы списка корутин
```

```
asyncio.gather()
```

Ключевые слова `async/await`

```
import asyncio
```

```
async def hello(name):  
    return "Hello, {}!".format(name)
```

<https://xakep.ru/2017/01/11/python-3-asyncio/>

Нам нужен event loop

```
import asyncio
```

```
async def hello(name):  
    return "Hello, {}!".format(name)
```

```
async def call_vasya():  
    greeting = await hello("Vasya")  
    return greeting
```

```
loop = asyncio.get_event_loop()  
print(loop.run_until_complete(call_vasya()))
```



Удобное решение задачи



<https://kevinmccarthy.org/2016/07/25/streaming-subprocess-stdin-and-stdout-with-asyncio-in-python/>



<https://github.com/aio-lib>

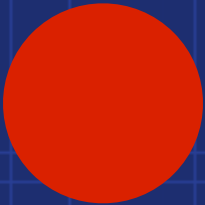
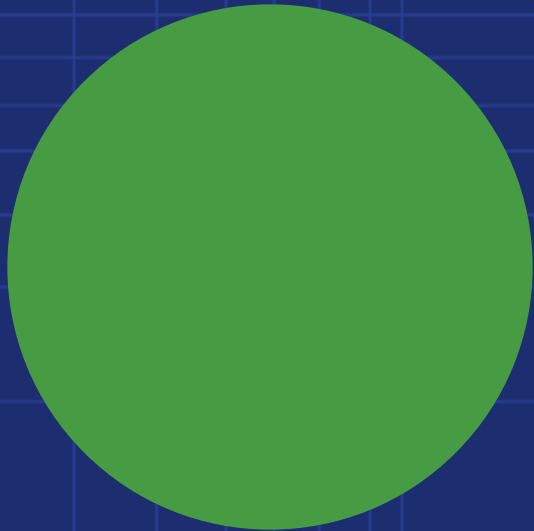
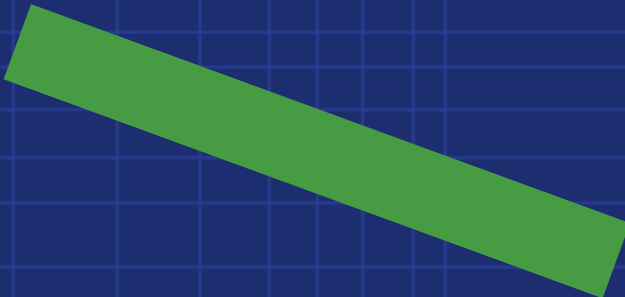


Упражнения

1. Напишите асинхронную реализацию `sleepsort`
2. Какой метод в `asynсіо` можно использовать для того, чтобы все-таки запускать поток ОС или процесс?



Асинхронные хранилища



Проблемы реляционных баз

- Не очень хорошо масштабируются
- Любое изменение схемы приводит к гиганским миграциям
- Плохо поддерживают асинхронность
- Распространенные СУБД плохо интегрируются с вычислительными решениями
- Но вообще PostgreSQL неплох

Как насчет NoSQL?

- Redis (<https://redis.io/>), <https://aioredis.readthedocs.io/en/latest/>
- Elasticsearch (<https://www.elastic.co/products/elasticsearch>), <https://aioes.readthedocs.io/en/latest/>
- MongoDB (<https://www.mongodb.com/>), <https://motor.readthedocs.io/en/stable/>

Elasticsearch как пример

```
pip install aioes
```

```
import asyncio
```

```
import aiohttp
```

```
from aioes import Elasticsearch  
from datetime import datetime
```

```
es = Elasticsearch(['localhost:9200'])
```



Elasticsearch как пример




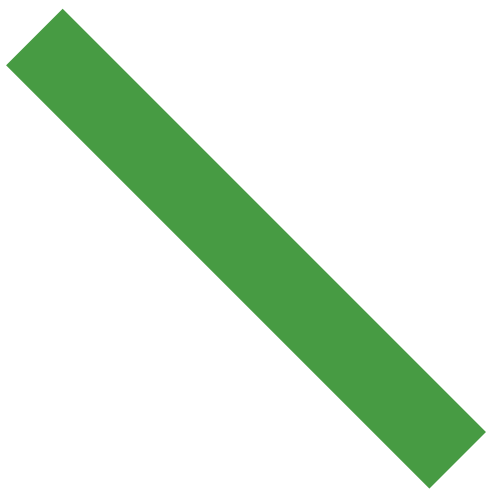
`pip install aioes`

`import asyncio`

`import aiohttp`

`from aioes import Elasticsearch`
`from datetime import datetime`

`es = Elasticsearch(['localhost:9200'])`



Elasticsearch как пример

URL = "http://bit.ly/2zsqrck"

```
async def create_db():
    async with aiohttp.ClientSession() as session:
        async with session.get(URL) as resp:
            films_urls = (await resp.json())["films"]

    for i, film_url in enumerate(films_urls):
        async with session.get(film_url) as resp:
            res = await es.index(
                index="coding-index",
                doc_type='film',
                id=i,
                body=await resp.json()
            )
            print(res['created'])
```

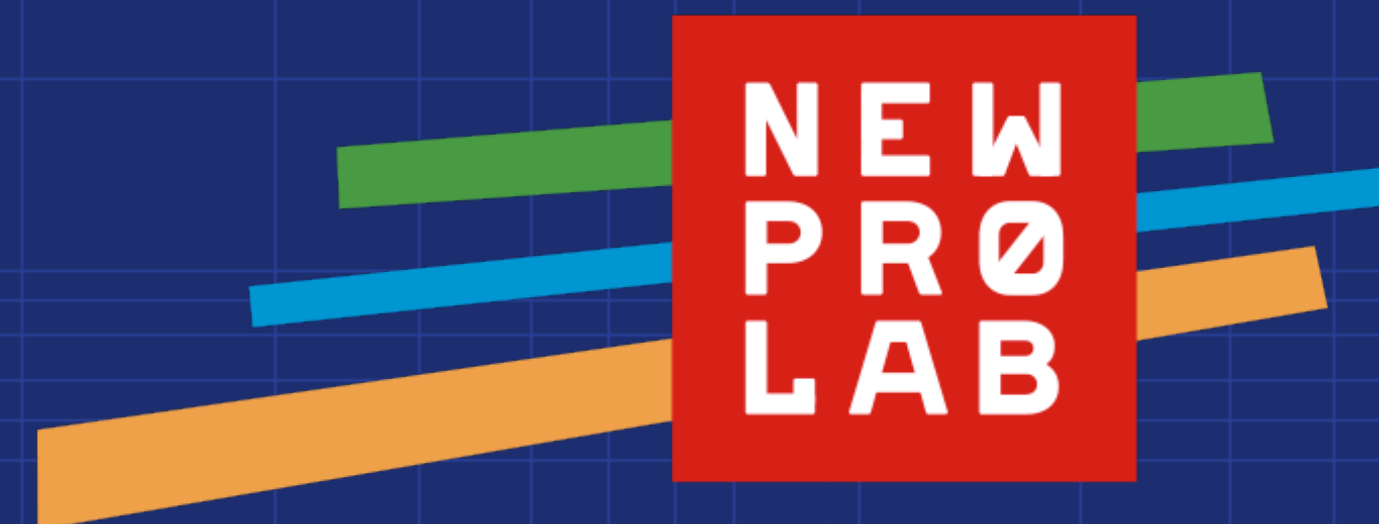
Elasticsearch как пример

<https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query.html>

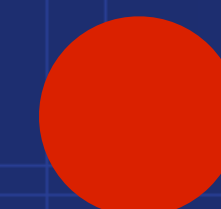
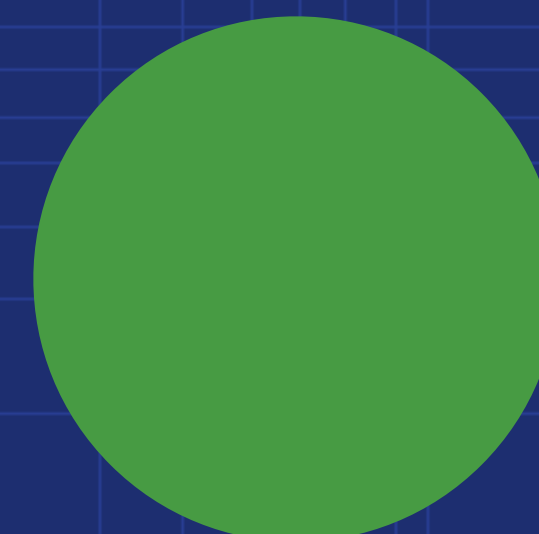
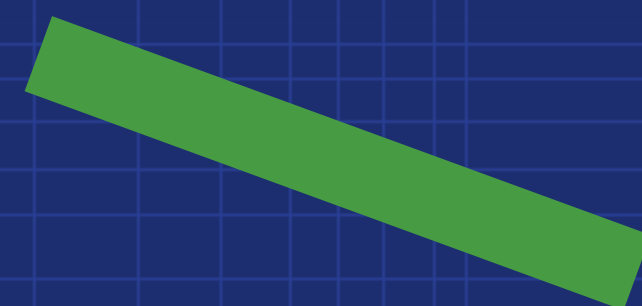
```
async def get_by_id(key):  
    return await es.get(index='coding-index', doc_type='film', id=key)  
  
async def search_by_director(director):  
    return await es.search(index='coding-index', body={"query": {"match": {'director': director}}})  
  
async def search_in_description(sentence):  
    return await es.search(index='coding-index', body={"query": {"match": {'description': sentence}}})
```

Elasticsearch как пример

```
loop = asyncio.get_event_loop()
loop.run_until_complete(create_db())
# loop.run_until_complete(get_by_id(0))
# loop.run_until_complete(search_by_director("Hayao Miyazaki"))
# loop.run_until_complete(search_in_description("cat"))
```



Микросервисы



Что такое микросервис?

Сервис, который имеет внешний API и решает одну конкретную независимую небольшую задачу.

<https://pastebin.com/p37Xf6jC>

aiohttp


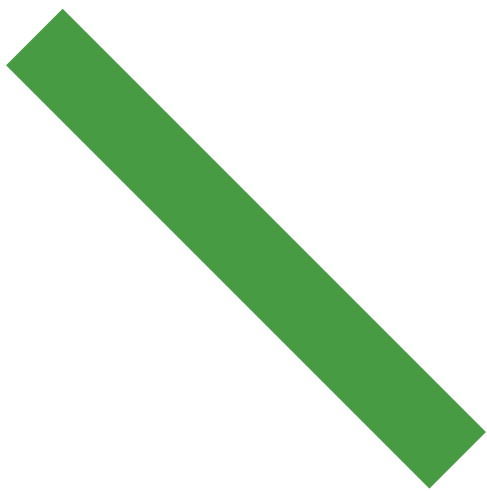


~\$ pip install aiohttp cchardet aiodns

<https://aiohttp.readthedocs.io/en/stable/>

Упражнение

Давайте реализуем аналог многопоточного кода по скачке файлов со слайда 8 на `aiohttp`. Какую проблему с асинхронностью нам при этом придется решить? Какой модуль поможет нам это сделать?



Больше практики!

Давайте скрестим сервер с клиентом и сделаем API:

- 1) Принимает URL на выбранный новостной сайт
- 2) Асинхронно достает текст статьи из него
- 3) Загружает текст и URL статьи в Elasticsearch
- 4) Позволяет кинуть в API два URL и найти сходство между документами по этим адресам