

# UDV Summer School 2021

## Разработка ПО



Спиненко Иван

Архитектор, ведущий разработчик





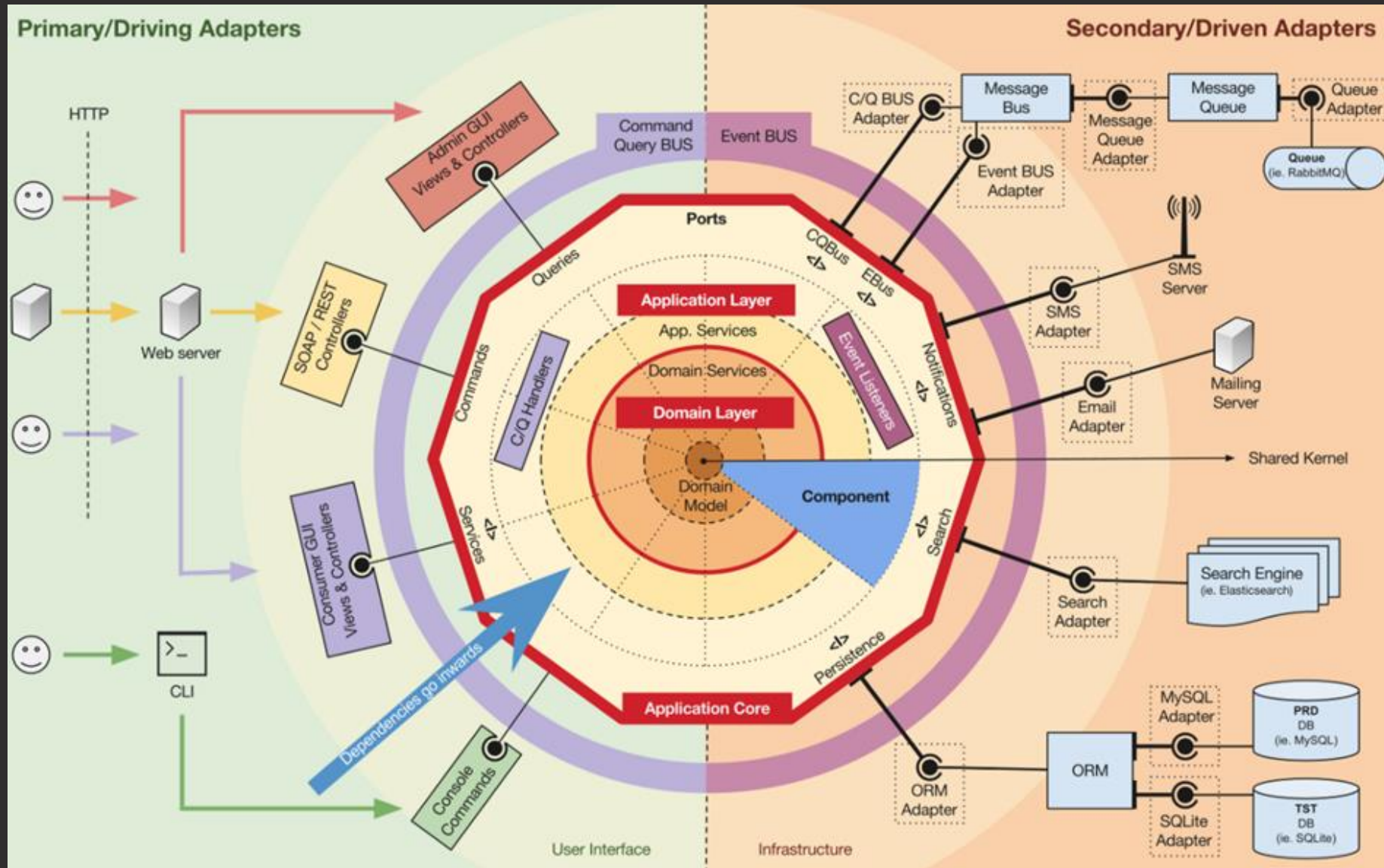
## Почему меняется подход к разработке ПО



- Все становится быстрее
- Многофункциональное программное обеспечение становится все более сложным
- Важны вопросы производительности, масштабируемости, планирования и распределения ресурсов, совместимости, лицензирования, длина жизненного цикла
- Существует множество технологий для решения задачи
- Для создания надежных и производительных архитектур необходим большой опыт и широкие знания: программирование, инструменты разработки, аппаратное обеспечение, технологии, ...



# Современная архитектура систем



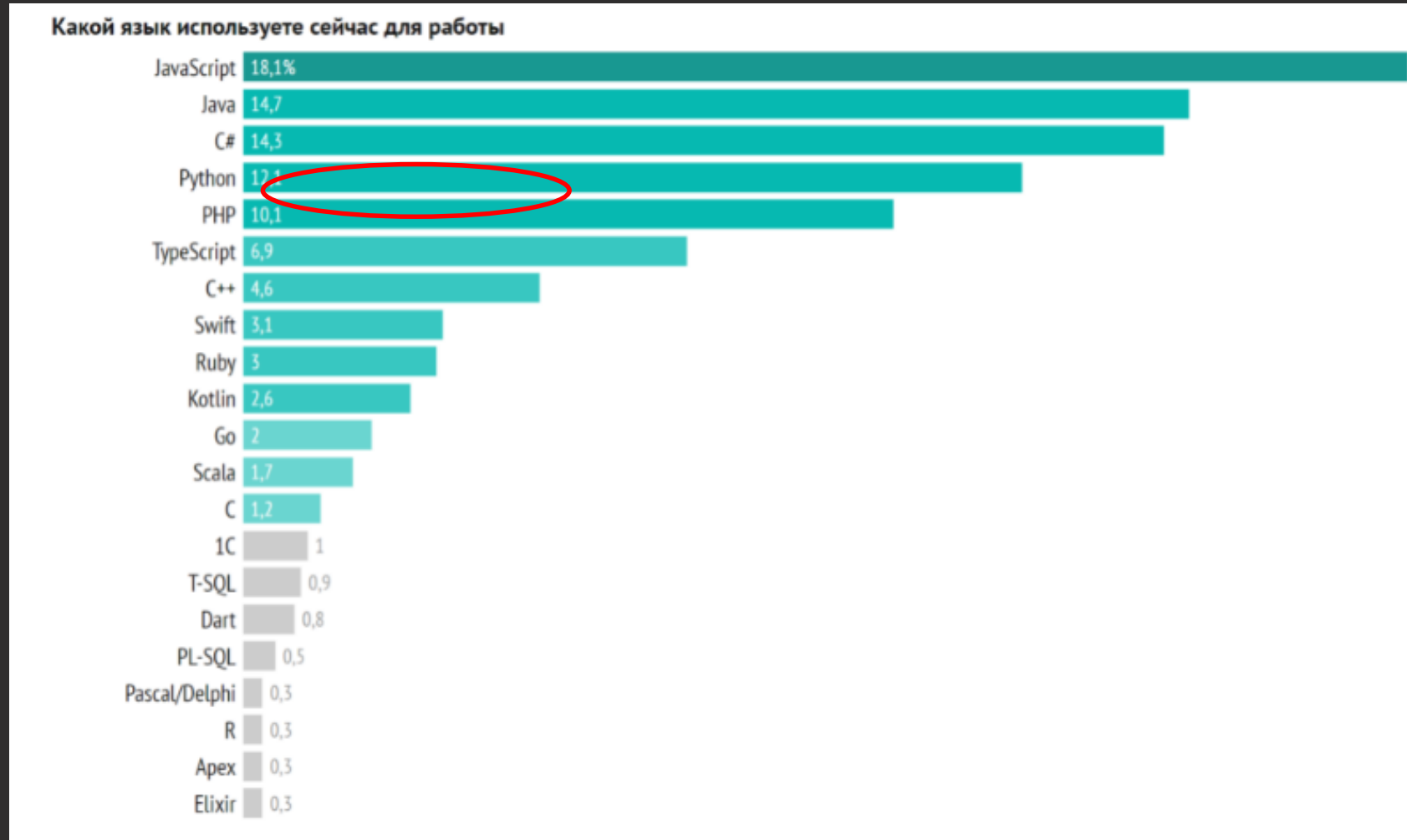
## > Различные языки программирования для разных систем



- Компилируемый / Интерпретируемый
- Быстрая и лаконичная разработка / Сложная структура и синтаксис
- Быстрое исполнение в run-time / Дополнительные издержки на выполнение операций «под капотом»
- Большое коммьюнити / Узкоспециализированный язык для небольшой группы специалистов
- Много open-source проектов / Закрытая коммерческая разработка

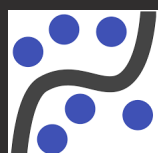


## Популярные языки программирования



<https://habr.com/ru/post/543346/>

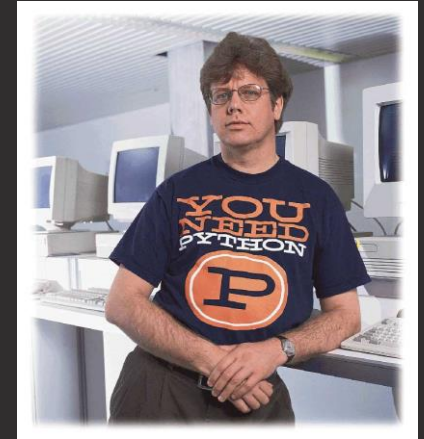




## > История



- Гвидо ван Россум — создатель, «великодушный пожизненный диктатор»
- Язык создан в 1991 году
- Интерпретируемый язык программирования
- ... (много всего интересного с 1991 год по наше время) ...
- Всплеск интереса сейчас: простой, наглядный, удобен для прототипирования, изучают даже не программисты (например, топ-менеджеры)
- PEP — Python Enhancement Proposal (предложения по развитию Python): это документ со стандартизированным дизайном, предоставляющий общую информацию о языке Python, включая новые предложения, описания и разъяснения возможностей языка.



## > Где используется



- Автоматизация рутинных операций
- Быстрое прототипирование (проверить гипотезу, алгоритм, метод из статьи)
- Когда скорость выполнения менее важна, чем читаемость и поддерживаемость кода
- Когда нет риска от использования различных open-source продуктов, сырых библиотек
- Инженерные расчеты, подготовка статей
- Работа с данными и машинное обучение



- Простота: можно быстро начать программировать без глубокого изучения синтаксиса
- Лаконичность: можно быстро запрограммировать идею
- Наличие мощных библиотек, созданных «коллективным разумом»
- Бесплатность
- Большое коммьюнити (много разработчиков, open-source проектов, чатов, форумов для обсуждения)
- Высокоуровневый язык: например, не нужно управлять памятью вручную
- Кроссплатформенность

- Не риалтайм
- GIL (неэффективная работа с потоками)
- Скорость выполнения программы ниже, чем на других языках (например, C)
- Нет компиляции, может упасть при выполнении (в run-time)
- Эволюция языка: частые изменения в самом языке и библиотеках проводят к несовместимости

## > Дзен python



- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один — и, желательно, только один — очевидный способ сделать это.
- Хотя он поначалу может быть и не очевиден, если вы не голландец
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем прямо сейчас.
- Если реализацию сложно объяснить — идея плоха.
- Если реализацию легко объяснить — идея, возможно, хороша.
- Пространства имён — отличная штука! Будем делать их побольше!

- Интерпретатор ([python.org](https://python.org))
- IPython (<https://ipython.org/>)
- Anaconda (<https://www.anaconda.com/>)
- Jupyter notebook, jupyterhub (<https://jupyter.org>)
- Binder (<https://mybinder.org/>)
- Google colab (<https://colab.research.google.com/>)
- Фреймворки разработки
- Django, Flask, aiohttp, TensorFlow, FastAPI, ...

## > Сущности языка (все есть объект)



- Числовые переменные
- Строки
- Списки
- Кортежи
- Словари
- Множества
- Функции
- Классы и объекты

```
[19] for elem in [1, 1., 1j, 1e-1, "one", True]:  
      print(type(elem))
```

```
<class 'int'>  
<class 'float'>  
<class 'complex'>  
<class 'float'>  
<class 'str'>  
<class 'bool'>
```

```
[30] a = []  
      def f():  
          return 123  
      a.append(f)  
      l = lambda : '123'  
      a.append(l)  
      a.append([1,2,3])  
      a.append((1,2,3,))  
      a.append("123")  
      a.append({'a':1, 'b':2, 'c':3})  
      a.append({1,2,3})  
      a.append((i for i in range(1,4)))  
      for i in a:  
          print(type(i), repr(i), i)
```

```
<class 'function'> <function f at 0x7f6ee7a9ca70> <function f at 0x7f6ee7a9ca70>  
<class 'function'> <function <lambda> at 0x7f6ee7a3c290> <function <lambda> at 0x7f6ee7a3c290>  
<class 'list'> [1, 2, 3] [1, 2, 3]  
<class 'tuple'> (1, 2, 3) (1, 2, 3)  
<class 'str'> '123' 123  
<class 'dict'> {'a': 1, 'b': 2, 'c': 3} {'a': 1, 'b': 2, 'c': 3}  
<class 'set'> {1, 2, 3} {1, 2, 3}
```



- Видимость переменных
- Параметры

```
[82] def fSquare(x):  
      return x**2  
      print(fSquare(10))  
      print([fSquare(i) for i in range(11)])  
      print(type(fSquare))
```

```
100  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]  
<class 'function'>
```

### Параметры функции

```
[83] def f(a, b=0):  
      print("a={}, b={}".format(a, b))  
      f(1, 2)  
      f(3)
```

```
a=1, b=2  
a=3, b=0
```

# > Операторы



- Условный оператор

```
[90] print("Укажите Ваш возраст:")
age = int(input())
if age < 6:
    print("Здравствуй, малыш")
elif age < 14:
    print("Привет")
elif age < 25:
    print("Добрый день")
elif age < 100:
    print("Здравствуйте")
else:
    print("Какой необычный возраст")
```

Укажите Ваш возраст:  
5  
Здравствуй, малыш

- Цикл while

```
[92] i, m = 0, []
while i < 5:
    m.append(i**2)
    i += 1
else:
    print(m)
```

[0, 1, 4, 9, 16]

- Цикл for

```
[93] for i in range(10):
    print(i**2, end=" ")
```

0 1 4 9 16 25 36 49 64 81

```
[94] for i in ["один", "два", "три"]:
    print(i)
```

один  
два  
три



## Работа с файлами



- Текстовый файл
- Json
- Xml
- Архивы
- Сокеты
- pipe

### Запись в файл

```
[102] with open('a.out', 'w') as f:  
      f.write("hello")  
      f.close()
```

### Чтение из файла

```
[103] with open('a.out', 'r') as f:  
      print(f.read())  
      f.close()
```

hello

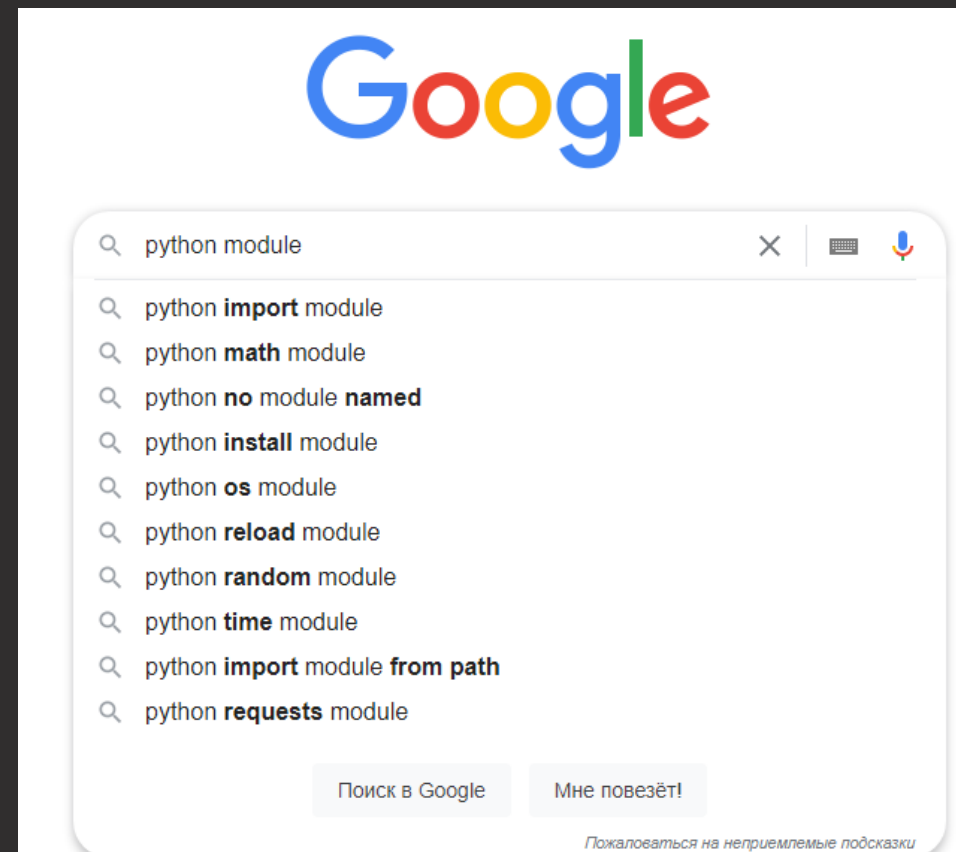
- `import * from ...`
- `from a import b as c`
- `__init__.py`

```
1 test/                                # Корневая папка
2     packA/                           # Пакет packA
3         subA/                        # Подпакет subA
4             __init__.py
5             sa1.py
6             sa2.py
7             __init__.py
8             a1.py
9             a2.py
10    packB/                            # Пакет packB (неявный пакет пространства имён)
11        b1.py
12        b2.py
13    math.py
14    random.py
15    other.py
16    start.py
```

## > Библиотеки (100500 штук)



- os
- sys
- time
  - json
- random
- collections
- matplotlib
- pandas
- numpy







## Автоматизация тестирования



- pytest

```
===== test session starts =====
platform darwin -- Python 3.8.5, pytest-6.1.1, py-1.9.0, pluggy-0.13.1
rootdir: /Users/benoit/Python/advanced-pytest-tricks
plugins: freezegun-0.4.2
collected 9 items

test_exception_example.py . [ 11%]
test_logging_example.py . [ 22%]
test_parametrized_example.py ..F... [ 88%]
test_time_example.py . [100%]

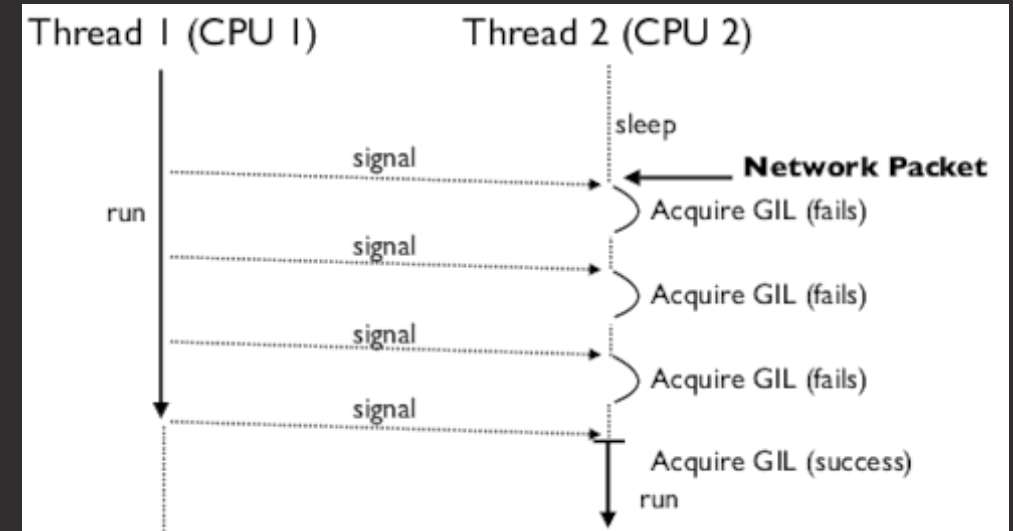
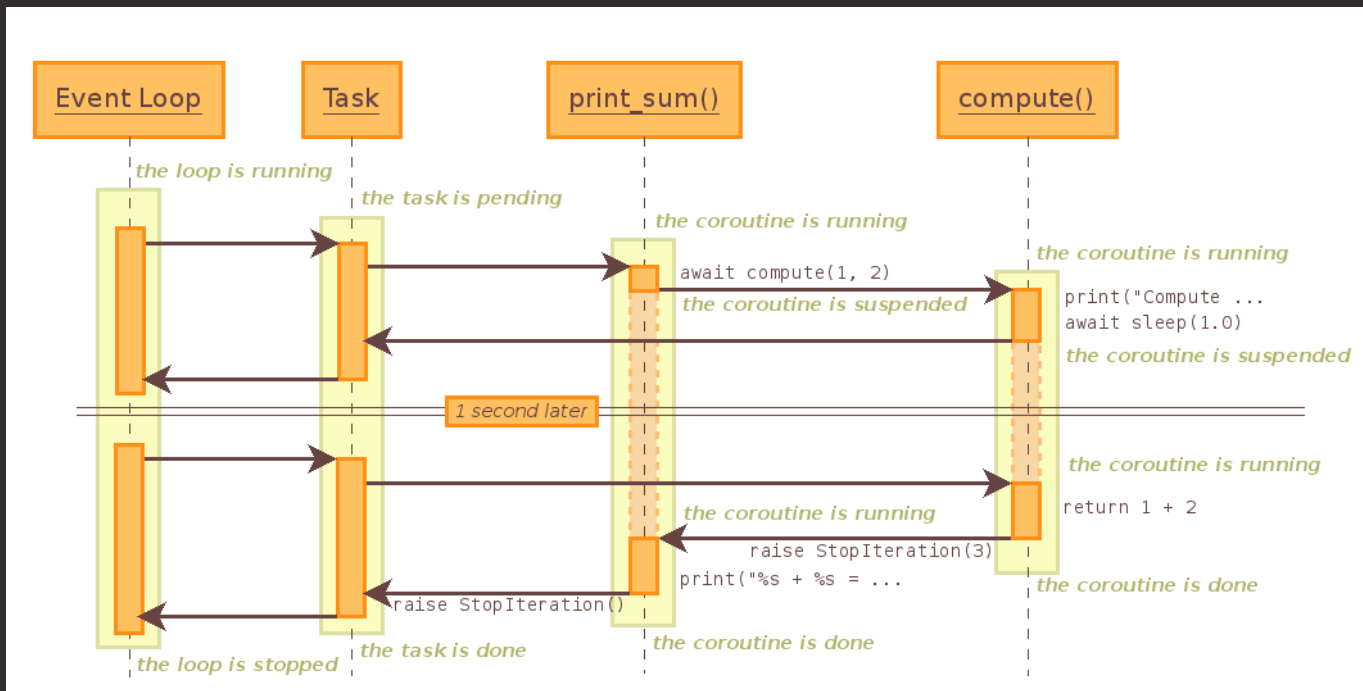
===== FAILURES =====
test_parametrized_example.py:18: in test_is_valid_email_address
    assert is_valid_email_address(test_input) == expected
E   AssertionError: assert False == True
E   + where False = is_valid_email_address('and-another@custom.org')

===== short test summary info =====
FAILED test_parametrized_example.py::test_is_valid_email_address[and-another@custom.org-True] - AssertionError: assert False == True
2560 x 648
===== 1 failed, 8 passed in 0.18s =====
```

## > Работа с потоками и процессами



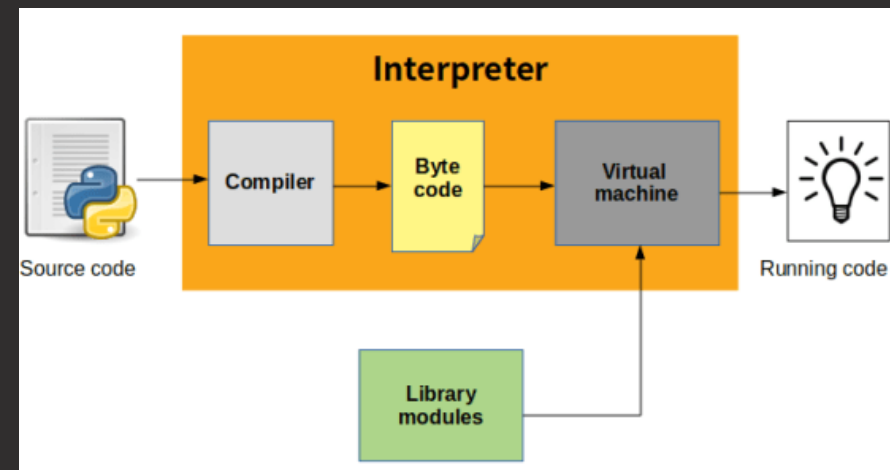
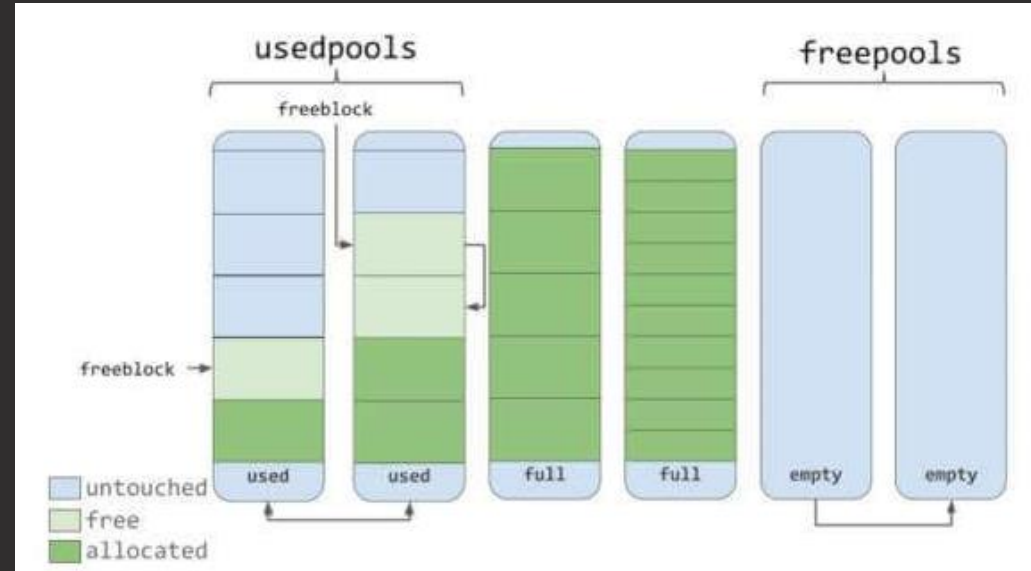
- multithreading / multiprocessing
- GIL
- asyncio



## > Низкоуровневый python



- Как работать с памятью
- Объект это ссылка
- Глубокое копирование
- Работа с мусором (import gc)
- CPython





## Полезные ссылки





- Официальный сайт python: <https://www.python.org/>
- Документация: <https://docs.python.org/3/>

### Примеры проектов


- <https://soshace.com/dockerizing-django-with-postgres-redis-and-celery/>
- <https://github.com/spinenkoia/mail-test>

### Программирование

- Лекции Т.Ф.Хирьянова «Алгоритмы и структуры данных на python»:  
<https://www.youtube.com/playlist?list=PLRDzFCPr95fK7tr47883DFUbm4GeOjjc0>
- Учебники: Кнут, Кормен
- Соревновательная платформа: RuCode



**Спасибо за внимание!**  
Вопросы?



Чернышов Юрий, [ychernyshov@ussc.ru](mailto:ychernyshov@ussc.ru), @yuchernyshov  
Астафьева Анна, @astafevaanny

