

# Углубленный Python

Лекция 1

Введение в Python, основные понятия

Тестирование

Кандауров Геннадий



образование

## Преподаватели



Геннадий Кандауров



Антон Кухтичев

# Напоминание отметиться на портале

+ ОСТАВИТЬ ОТЗЫВ

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly course schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

Below the schedule, there's a blog post titled 'Углубленный Python' (Advanced Python) with the subtitle 'Blog for materials on the "Advanced Python" course'. It has 57 readers and 2 topics. A search bar and a 'Найти' (Find) button are below the post. To the right, there's a sidebar for 'Прямой эфир' (Live Stream) and a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

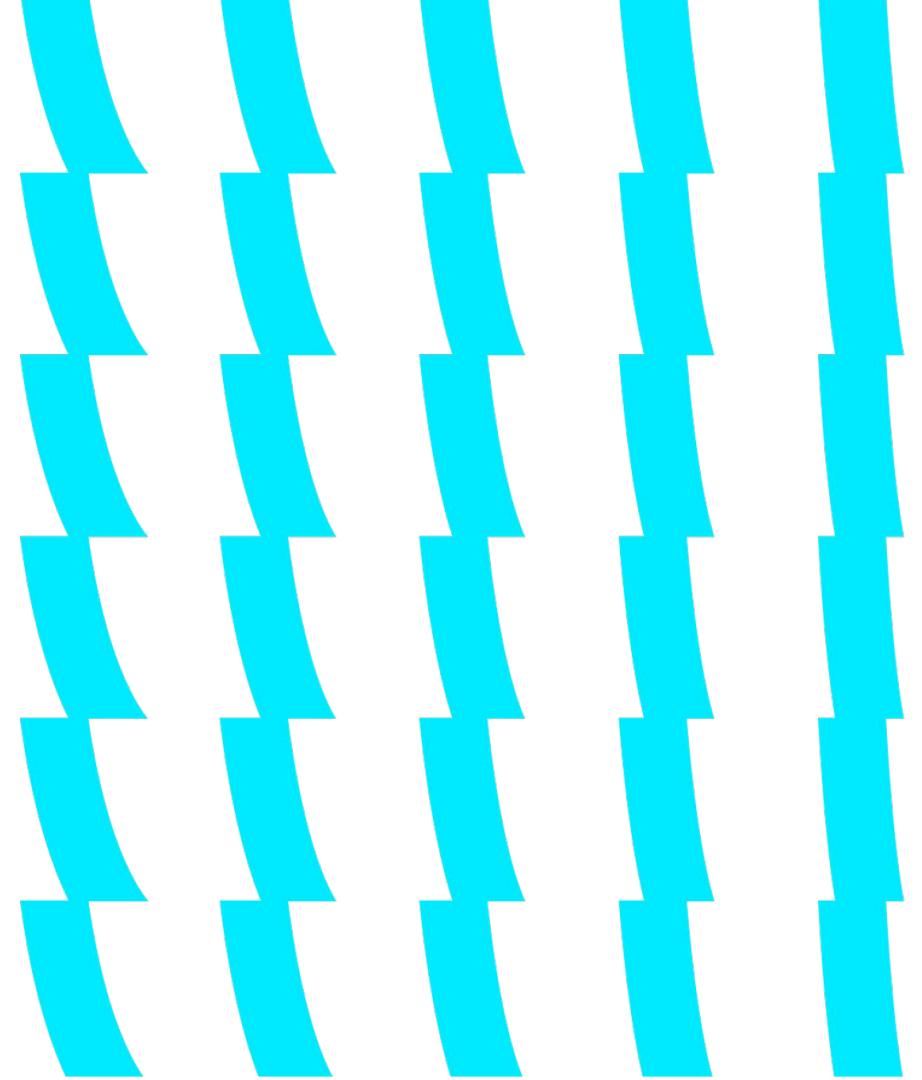
## План курса

1. Введение в Python, тестирование
2. Функции, функциональное программирование
3. Объектная модель, классы
4. ООП, метaprogramмирование
5. Стандартная библиотека
6. Потоки, GIL, процессы, IPC
7. Асинхронное программирование
8. Устройство памяти, профилирование
9. Логирование, отладка, оформление кода
10. Расширения на C
11. Экзамен/пересдача

# Содержание занятия

1. Основные понятия
2. Типы данных
3. Управляющие конструкции
4. Функции
5. Классы
6. Тестирование

# Python



# Python

Интерпретируемый язык с динамической и утиной типизацией, автоматической сборкой мусора и GIL.

Python - название спецификации языка

Реализации:

- CPython
- IronPython
- Jython
- PyPy

<https://github.com/python/cpython>

<https://www.python.org/doc/>



# Дзен Python

```
import this
```

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

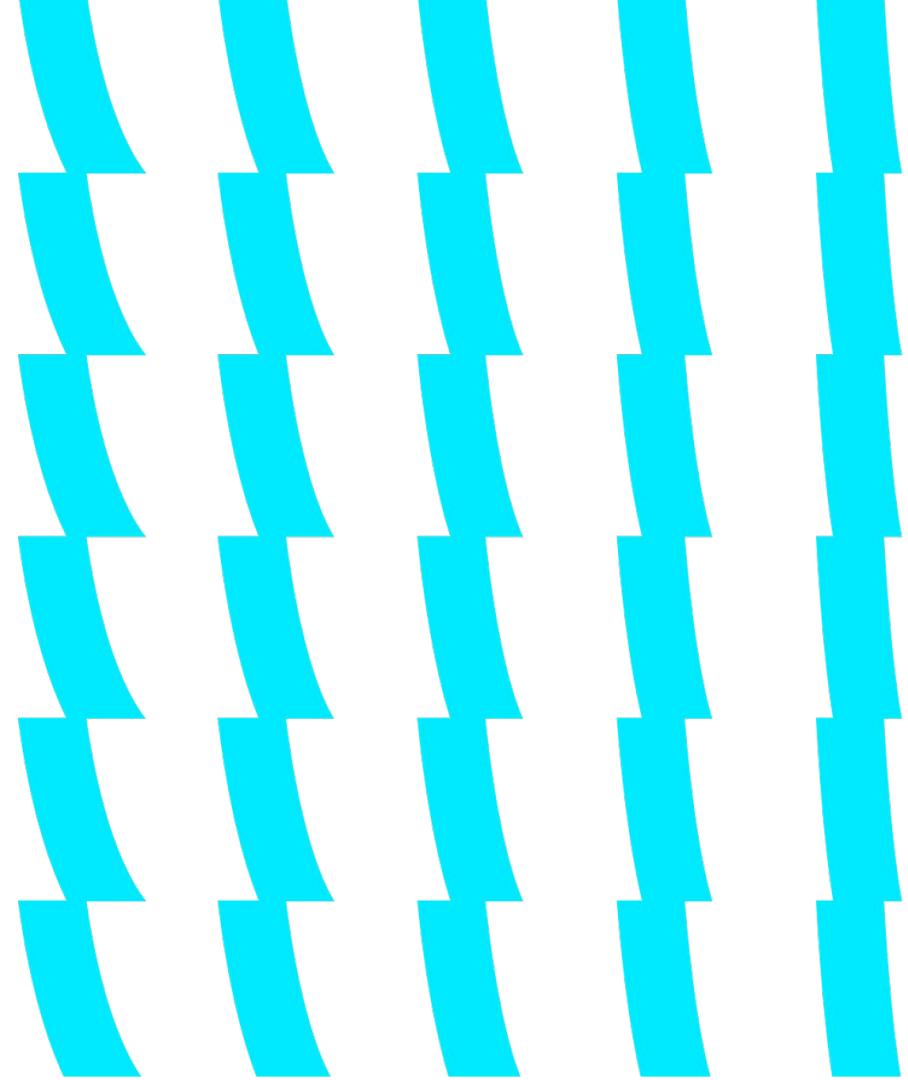
Although never is often better than \*right\* now.

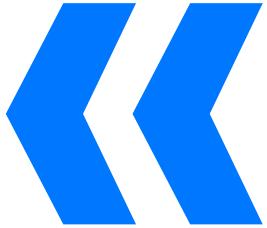
If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

**Все есть объект**





*Objects are Python's abstraction for data. All data in a Python program is represented by objects or by relations between objects.*

[docs.python.org](https://docs.python.org)

# Объект

1. Каждый объект имеет идентичность, тип, значение
2. `id` никогда не меняется после создания объекта (`is` сравнивает `id` объектов)
3. Тип объекта определяет какие операции с ним можно выполнять
4. Значение объекта может меняться

# PyObject

```
typedef struct _object {
    _PyObject_HEAD_EXTRA
    Py_ssize_t ob_refcnt;
    PyTypeObject *ob_type;
} PyObject;
```

# Переменные

```
num = 1 # операция присваивания переменной num значения 1
```

- допустимые символы: буквы, цифры, \_ (подчеркивание)
- начинаться только с буквы или \_

# valid	# invalid
num = 1	1num = 1
_num = 1	1_num = 1
__num = 1	
число1_2_34 = 1	
_123_число = 1	

## Базовые типы: типы с одним значением

- `None`
- `NotImplemented`
- `Ellipsis (...)`

# Базовые типы: числа

## Целые числа (int)

```
num = 42  
num = 42_000_000  
num = 0o10 # 8  
num = 0x10 # 16
```

## Вещественные числа (float)

```
num = 3.14
```

## Комплексные числа (complex)

```
num = 9 + 0.75j # num.real, num.imag
```

## Арифметические операции

- Сложение: +
- Вычитание: -
- Умножение: \*
- Деление: /
- Целочисленное деление: //
- Остаток от деления: %
- Возведение в степень: \*\*

## Битовые операции

- Умножение: &
- Сложение: |
- Инверсия: ~
- Исключающее ИЛИ: ^
- Побитовый сдвиг: <<, >>

# Базовые типы: bool

## Логический тип (bool)

```
is_true = True # или False
```

- Логическое И: **and**
- Логическое ИЛИ: **or**
- Отрицание: **not**

Вычисление составных логических выражений **ленивое**:

```
return_false() and return_true()
```

```
return_true() or return_false()
```

# Базовые типы: строки

```
s = "просто строка" # str  
byte_s = b"qwerty" # bytes  
raw_s = r"111\nданные" # str  
  
str.encode() -> bytes  
bytes.decode() -> str  
  
str1 + str2 # конкатенация  
"a" * 5 # повторение "aaaaa"
```

## Форматирование

- "x %s %d" % ("y", 42)
  - "x %(y)s %(n)d" % dict(y="y", n=42)
  - "x {} {}".format("y", 42)
  - y, n = "y", 42
- f"x {y} {n}"
- "x y 42"

# Базовые типы: коллекции

## Список (list)

```
lst = []
lst = [1, 2, 3]
lst1 = list(lst)
lst.append(4)
```

## Кортеж (tuple)

```
tup = ()
tup = (1,)
tup = 1, 2, 3
tup = tuple(lst)
```

## Swap

```
a, b = b, a
```

## Распаковка

```
lst = [1, 2, 3, 4, 5]
a, b, *c = lst # 1, 2, [3, 4, 5]
a, *b, c = lst # 1, [2, 3, 4], 5
```

```
lst = ["123", "456"]
(a, *b), c = lst
# "1", ["2", "3"], "456"
```

# Базовые типы: коллекции

## Словарь (dict)

```
d = {} # dict()  
d1 = {1: 11}  
d1[2] = 22  
d2 = dict(x=10, y=20)  
d3 = {**d1, **d2, 3: 33}
```

**Множество (set, frozenset)** уникальных неизменяемых объектов.  
Множество не индексируется, но по нему можно итерироваться.

```
s = set()  
s = {1, 2, 3}  
s.add(4)  
s = set("123123123") # {"1", "2", "3"}
```

# Базовые типы: коллекции

```
lst = [1, 3, 2]
```

Списковые включения (компрехеншены)

```
dct = {1: 11, 2: 22}
```

```
lst = [n ** 2 for n in range(10) if n % 2]
```

- Оператор in:

```
1 in lst
```

```
s = {ch for ch in "abcabcbca"}
```

```
1 in dct
```

```
dct = {n: n ** 2 for n in range(10) if n % 2}
```

```
"456" in "123456"
```

- Сортировка:

```
sorted(dct)
```

```
gen = (n ** 2 for n in range(10) if n % 2)
```

```
lst.sort()
```

- Итерирование:

```
for key in dct: pass
```

# Изменяемые, неизменяемые типы

## Неизменяемые

- int, float, bool, complex
- str, bytes
- tuple
- frozenset

## Изменяемые

- list
- dict, set
- user defined

# Управляющие конструкции: циклы

## Цикл `for`

```
for elem in sequence:  
    process(elem)  
    # break  
    # continue  
  
else:  
    do_else()
```

## Цикл `while`

```
while condition:  
    process()  
    # break  
    # continue  
  
else:  
    do_else()
```

# Управляющие конструкции: условный оператор

```
if cond1:  
    action1()  
  
elif cond2:  
    action2()  
  
else:  
    action3()
```

## Тернарный оператор

```
result = action1() if cond else action2()
```

## Pattern matching (3.10)

```
value = 42  
  
match value:  
    case 41:  
        act_41()  
  
    case 42:  
        act_42()  
  
    case other:  
        act_other(other)
```

# Управляющие конструкции: контекстный менеджер

```
class CtxManager:  
    def __init__(self, name):  
        self.db = connect_db(name)  
    def __enter__(self):  
        return self.db  
    def __exit__(self, exc_type, exc_val, exc_tb):  
        self.db.close()  
  
# from contextlib import contextmanager  
  
with CtxManager("db_name") as db:  
    do_action(db)
```

# Управляющие конструкции: исключения

```
try:  
    something_dangerous()  
except DangerError as err:  
    process_danger_error(err)  
except TrivialError as err:  
    process_trivial_error(err)  
except Exception as err:  
    process_total_error(err)  
else:  
    process_no_error()  
finally:  
    will_be_executed_in_any_case()
```

# Управляющие конструкции: исключения

```
# правильно наследоваться от Exception
class TrivialError(Exception): pass

class DangerError(TrivialError): pass

# плохо, только когда реально надо
class UserError(BaseException): pass

# совсем плохо
try:
    pass
except:
    process_err()
```

# Функции, lambda-функции

```
def add(a, b):
    return a + b

def do_action(action, *args, **kwargs):
    print(f"do {action} with {args}, {kwargs}")

    if callable(action):
        return action(*args, **kwargs)
    else:
        return action

do_action(add, 1, b=2)
do_action(lambda x, y: x * y, 5, y=6)
```

# ФУНКЦИИ: ЗАМЫКАНИЕ

**Замыкание** - функция первого класса, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся её параметрами.

```
def get_summator(n):  
    def add(m):  
        return n + m  
    return add  
  
add2 = get_summator(2)  
add42 = get_summator(42)  
add2(5) # 7  
add42(5) # 47
```

# ФУНКЦИИ: декораторы

**Декоратор** - это функция, которая принимает функцию и возвращает функцию.

```
def deco(fn):  
    def inner(*args, **kwargs):  
        print("before", fn.__name__)  
        res = fn(*args, **kwargs)  
        print("after", fn.__name__)  
        return res  
    return inner
```

```
@deco
```

```
def add_nums(a, b):  
    return a + b
```

# ФУНКЦИИ: декораторы с параметрами

```
def deco(add_param):  
    def inner_deco(fn):  
        def inner(*args, **kwargs):  
            return fn(*args, **kwargs) + add_param  
        return inner  
    return inner_deco  
  
@deco(45)  
def add_nums(a, b):  
    return a + b
```

# Итераторы

**Итератор** представляет собой объект-перечислитель, который для данного объекта выдает следующий элемент, либо вызывает исключение, если элементов больше нет.

```
num_list = [1, 2, 3]
itr = iter(num_list)
print(next(itr)) # 1
print(next(itr)) # 2
print(next(itr)) # 3
print(next(itr)) # StopIteration
```

```
class SpecialIterator:
    def __init__(self, limit):
        self.limit = limit
        self.counter = 0
    def __next__(self):
        if self.counter < self.limit:
            self.counter += 1
            return self.counter
        else:
            raise StopIteration
```

```
iter_obj = SpecialIterator(3)
print(next(iter_obj))
```

# ФУНКЦИИ: генераторы

Генератор – функция, которая при вызове `next()` возвращает следующий объект по алгоритму ее работы. Вместо `return` в генераторе используем `yield` (или вместе).

```
def gen(count):
    while count > 0:
        yield count
        count -= 1
    return count # будет аргументом StopIteration

for i in gen(5):
    print(i) # 5, 4, 3, 2, 1
```

# Классы

```
class A:  
    cls_attr = 42  
  
    def __init__(self, val):  
        self.val = val  
        self._protected = 10  
        self.__private_val = 20  
  
    def print_name(self):  
        print(self.__class__.__name__)  
  
    @property  
    def private_val(self):  
        return self.__private_val
```

```
class B(A):  
    cls_attr = 55  
  
    def __init__(self, bval, val):  
        super().__init__(val)  
  
        self.bval = val  
        self.__private_val = 999  
  
    b = B(1, 2)  
    print(b.private_val) # ???  
  
    isinstance(b, (int, A)) # ???  
    issubclass(B, object) # ???
```

# Модули и пакеты

Модули являются основным компонентом организации кода в питоне (и это тоже объекты), объединяются в пакеты.

```
import this
collections = importlib.import_module('collections')
itertools = __import__('itertools')
```

file.py - модуль

--app/ - пакет

|--\_\_init\_\_.py

|--hello.py

## Валидные импорты

```
import sys
from sys import path
from sys import *
import sys as s
from sys import path as sys_path
```

# **Работа с файлами**

# Типы операций с файлами

- связанные с его открытием: открытие, закрытие файла, запись, чтение, перемещение по файлу и др.
- выполняющиеся без его открытия: работа с файлом как элементом файловой системы - переименование, копирование, получение атрибутов и др.

# Файловый объект

При открытии файла операционная система возвращает специальный дескриптор файла (идентификатор), однозначно определяющий, с каким файлом далее будут выполняться операции.

В Python работа с файлами осуществляется через специальный абстрактный файловый объект. В зависимости от способа создания такого объекта, он может быть привязан как к физическому файлу на диске, так и другому устройству, поддерживающему схожие операции (стандартный ввод/вывод и пр.).

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
newline=None, closefd=True, opener=None)
```

```
# кодировка  
import locale  
locale.getpreferredencoding(False)
```

# Файловый объект

```
f = open("some.file", "r")
data = f.read()
f.close()

# лучше
with open("some.file", "r") as f:
    data = f.read()
```

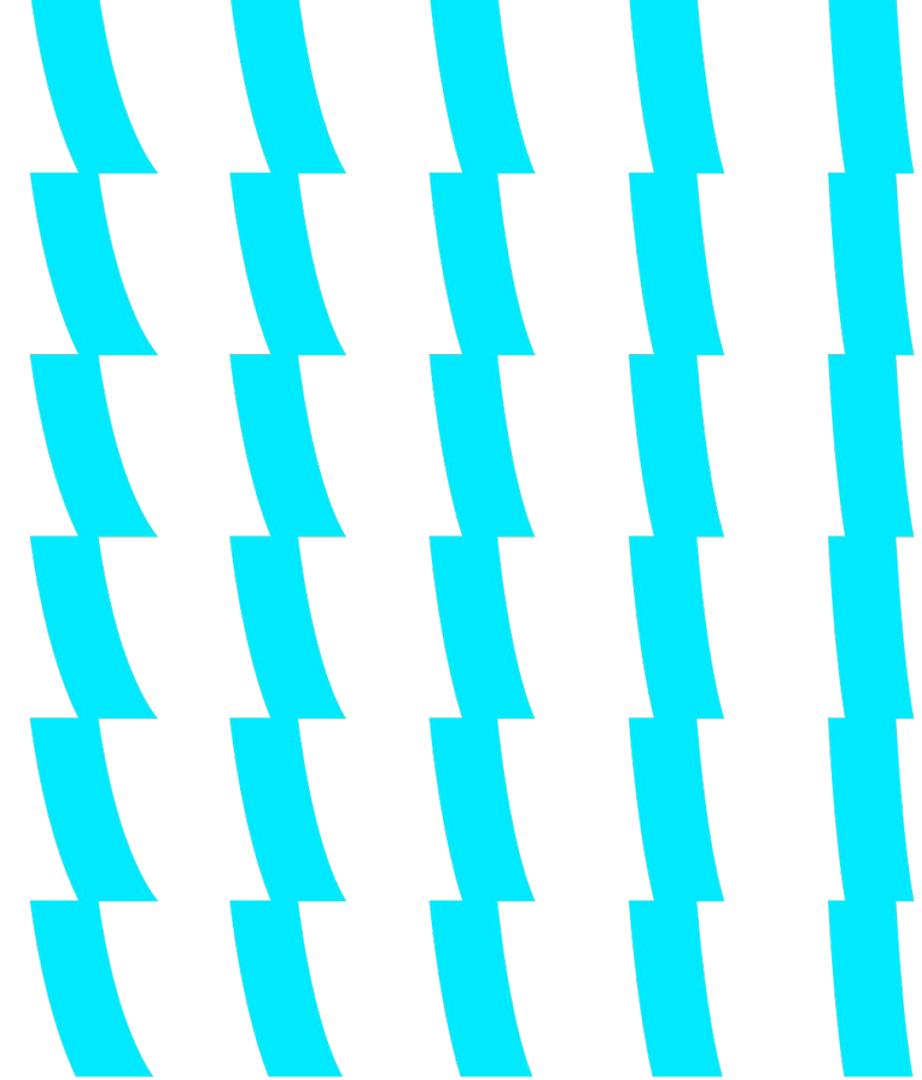
# Обработка файла

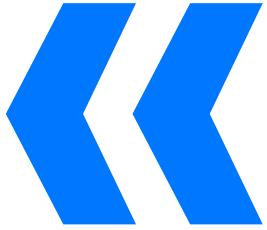
Режим	Описание
r	Только для чтения.
w	Только для записи. Создаст новый файл, если не найдет с указанным именем.
rb	Только для чтения (бинарный).
wb	Только для записи (бинарный). Создаст новый файл, если не найдет с указанным именем.
r+	Для чтения и записи.
rb+	Для чтения и записи (бинарный).
w+	Для чтения и записи. Создаст новый файл для записи, если не найдет с указанным именем.
wb+	Для чтения и записи (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
a	Откроет для добавления нового содержимого. Создаст новый файл для записи, если не найдет с указанным именем.
a+	Откроет для добавления нового содержимого. Создаст новый файл для чтения записи, если не найдет с указанным именем.
ab	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
ab+	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для чтения записи, если не найдет с указанным именем.

## Файловый объект: методы

- `file.close()`
- `file.fileno()`
- `file.flush()`
- `file.isatty()`
- `file.next()`
- `file.read(n)`
- `file.readline()`
- `file.readlines()`
- `file.seekable()`
- `file.tell()`
- `file.truncate(n)`
- `file.write(str)`
- `file.writelines(sequence)`

# Тестирование





*Тестирование показывает  
присутствие ошибок, а не их  
отсутствие.*

Эдсгер Дейстра

Тестирование можно доказать неправильность программы, но нельзя доказать её правильность.

## Цели тестирования

- Проверка правильности реализации
- Проверка обработки внештатных ситуаций и граничных условий
- Минимизация последствий
- Подготовка ко внесению изменений

# Виды тестирования

- Unit-тесты (модульные тесты)
- Функциональное тестирование
- Системное тестирование
- Интеграционное тестирование
- Регрессионное тестирование
- Тестирование производительности
  - Нагрузочное
  - Стress

## TDD

**TDD** (Test Driven Development) – техника разработки ПО, основывается на повторении коротких циклов разработки: пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и далее проводится рефакторинг нового кода.

## Степень покрытия тестами

coverage - библиотека для проверки покрытия тестами.

```
pip install coverage
```

```
coverage run tests.py
```

```
coverage report -m
```

```
coverage html
```

# Инструменты тестирования в Python

- doctest
- unittest
- pytest
- factory\_boy
- selenium

## doctest

```
def multiply(a, b):
    """
    >>> multiply(4, 3)
    12
    >>> multiply("a", 3)
    'aaa'
    """
    return a * b
```

```
python -m doctest <file>
```

# unittest

```
class TestCase
```

- `def setUp(self):`  
установки запускаются перед каждым тестом
- `def tearDown(self):`  
очистка после каждого метода
- `def test_<название теста>(self):`  
код теста

## unittest: TestCase

```
import unittest

class TestString(unittest.TestCase):

    def test_upper(self):

        self.assertEqual("text".upper(), "TEXT")

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

## unittest: набор assert\*

- assertEquals(a, b)
- assertNotEqual(a, b)
- assertTrue(x)
- assertFalse(x)
- assertIsNone(x)
- assertIs(a, b)
- assert IsNot(a, b)
- assertIn(a, b)
- assertIsInstance(a, b)
- assertLessEqual(a, b)
- assertListEqual(a, b)
- assertDictEqual(a, b)
- assertRaises(exc, fun,  
                  \*args, \*\*kwargs)

## unittest: mock

**Mock** – это объект-пустышка, который заменяет некий реальный объект (функцию, экземпляр) для определенной части программы.

- Высокая скорость
- Избежание нежелательных побочных эффектов во время тестирования
- Позволяет задать специальное поведение в рамках теста

```
from unittest.mock import patch

class TestUserSubscription(TestCase):
    @patch("users.views.get_status", return_value=True)
    def test_subscription(self, get_status_mock):
        ...
```

## unittest: mock

Атрибуты объекта Mock с информацией о вызовах

- `called` — вызывался ли объект вообще
- `call_count` — количество вызовов
- `call_args` — аргументы последнего вызова
- `call_args_list` — список всех аргументов
- `method_calls` — аргументы обращений к вложенным методам и атрибутам
- `mock_calls` — то же самое, но в целом и для самого объекта, и для вложенных

```
self.assertEqual(get_subscription_status_mock.call_count, 1)
```

## unittest: запуск тестов

```
# Найти и выполнить все тесты
```

```
python -m unittest discover
```

```
# Тесты нескольких модулей
```

```
python -m unittest test_module1 test_module2
```

```
# Тестирование одного кейса - набора тестов
```

```
python -m unittest tests.SomeTestCase
```

```
# Тестирование одного метода
```

```
python -m unittest tests.SomeTestCase.test_some_method
```

## [factory\\_boy](#)

Библиотека `factory_boy` служит для генерации разнообразных объектов (в т.ч. связанных) по заданным параметрам.

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

<https://faker.readthedocs.io/en/master/>

```
pip install factory_boy
```

# selenium

**Selenium** WebDriver – это программная библиотека для управления браузерами. WebDriver представляет собой драйверы для различных браузеров и клиентские библиотеки на разных языках программирования, предназначенные для управления этими драйверами.

```
pip install selenium
```

# selenium

- Требует конкретного драйвера для конкретного браузера (Chrome, Firefox и т.д.)
- Автоматическое управление браузером
- Поддержка Ajax
- Автоматические скриншоты

## Домашнее задание #1

- Реализовать игру крестики-нолики
- Написать тесты (`unittest` или `assert`)
- Проверить и поправить код `flake8` и `pylint`
- Проверить покрытие тестов через `coverage`

# Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

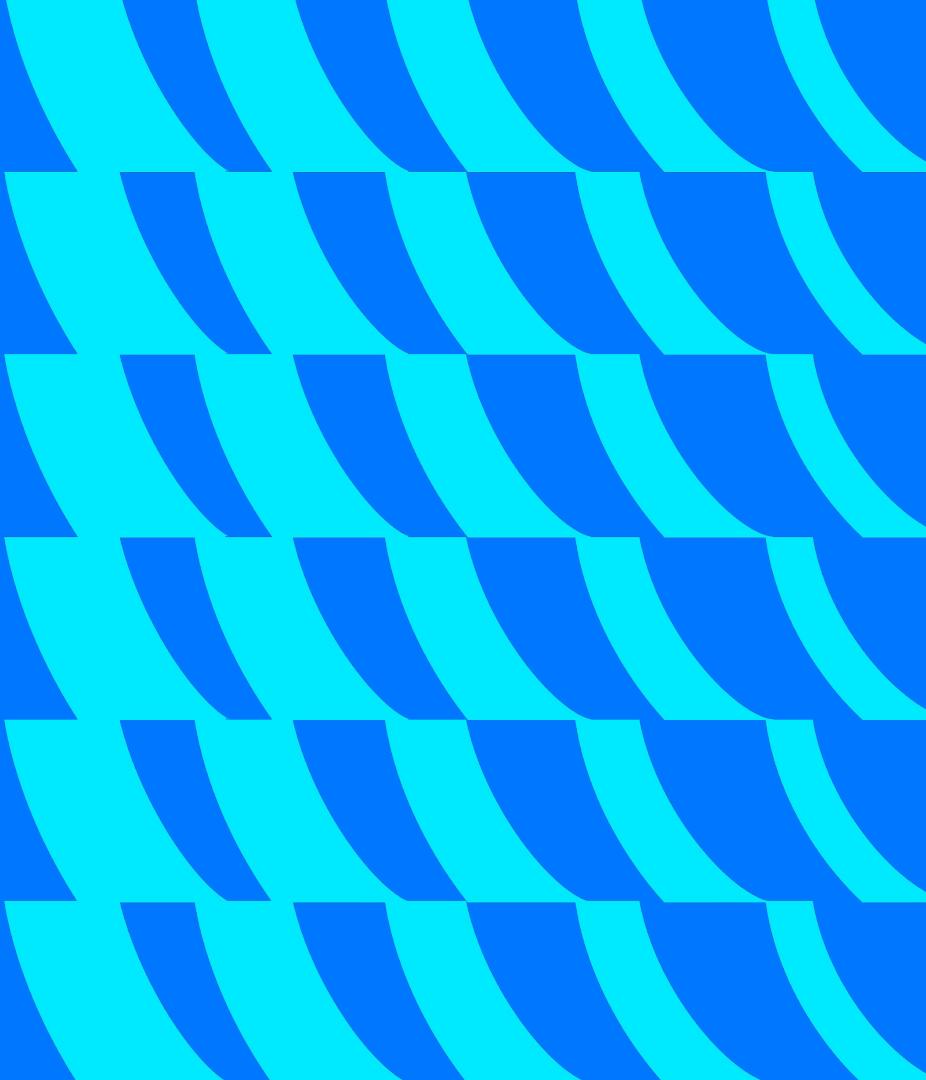
Below the schedule, a blog post titled 'Углубленный Python' is shown. It has 57 readers and 2 topics. There are buttons for 'Подписаться' (Subscribe) and 'Создать топик' (Create topic). A search bar and a 'Найти' (Find) button are also present.

On the right side, there's a sidebar for 'Прямой эфир' (Live Stream) with a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

Спасибо за  
внимание



образование



# Функции, встроенные функции

Антон Кухтичев

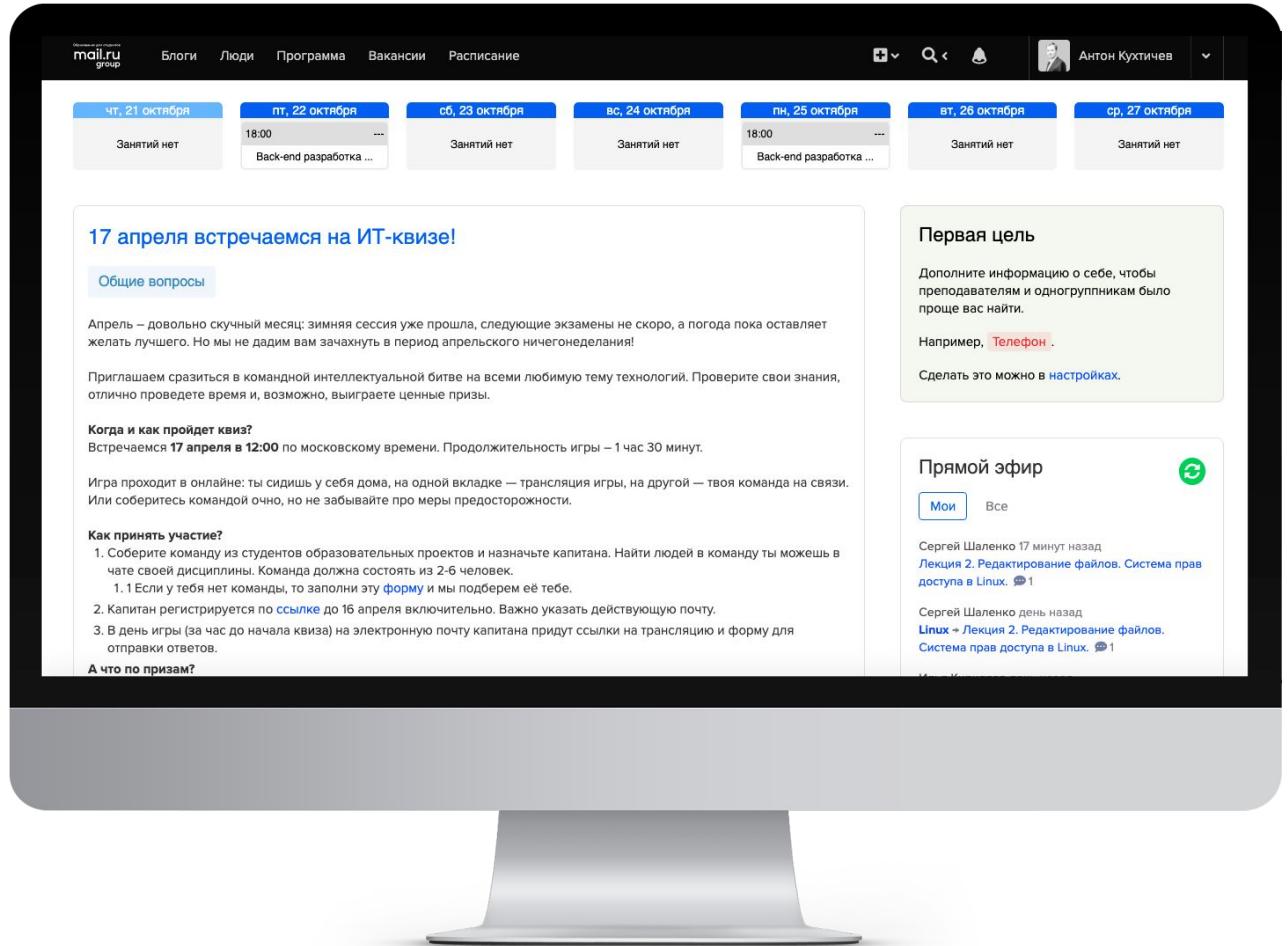


## Содержание занятия

- Квиз
- Функции
- Аргументы функции
- Декораторы
- λ-функция
- Встроенные функции

# Напоминание отметиться на портале

## и оставить отзыв после лекции



# Квиз #1

По первой лекции

<https://forms.gle/dyg97AaDXYYec7yV6>

# Функции и их аргументы

# ФУНКЦИИ

```
def square(x):  
    return x * x
```

```
>>> square(4)
```

```
16
```

Правила наименования:

- имя функции состоит из букв, чисел, знака подчёркивания (\_);
- название функции не должно начинаться с цифры;
- лидирующий знак подчёркивания - соглашение, что функция приватная.

# Аргументы функции

```
>>> def func(a, b, c=2): # c - необязательный аргумент  
...     return a + b + c
```

- может принимать произвольное количество аргументов;
- а может и не принимать их вовсе;
- может принимать и произвольное число именованных аргументов;
- у аргументов может быть значение по умолчанию;

# Декоратор

```
def square(func):  
    def wrapper(*args, **kwargs):  
        # какая-то логика  
        return wrapper
```

Декоратор - это функция, принимающая единственный аргумент - другую функцию и выполняющая дополнительную логику.

# λ-функция

```
f = lambda x: x * x
```

```
>>> f(2)
```

```
4
```

- Работают быстрее классических функций;
- Полезны в случае, когда нужна одноразовая функция;
- Потенциально повышают читаемость кода, а могут понизить.

## ФУНКЦИИ: параметры

```
def fn1(x, y=100): pass  
  
def fn2(*args): pass  
  
def fn3(**kwargs): pass  
  
def fn4(*args, **kwargs): pass  
  
def fn5(*, val): pass  
  
def fn6(start, stop, /): pass  
  
def fn7(pos1, /, pos2, pos3=3, *, kw1=11, **kwargs): pass
```

# ФУНКЦИИ: параметры

```
def make_function(name, *args, kw=12, **kwargs):  
    '''makes inner function'''  
  
    def inner(age=999):  
        print(f'{name=}, {age=}, {kw=}, {args=}, {kwargs=}')  
  
    return inner  
  
fn = make_function('skynet', 54, aim='term')  
  
fn()  
  
# name='skynet', age=999, kw=12, args=(54,), kwargs={'aim': 'term'}
```

# ФУНКЦИИ

```
>>> fn.__dict__
```

```
{}
```

```
>>> fn.music = 'yes'
```

```
>>> fn.__dict__
```

```
{'music': 'yes'}
```

```
>>> fn.music
```

```
'yes'
```

# ФУНКЦИИ: атрибуты

**`__doc__`** докстринг, изменяемое

```
>>> make_function.__doc__
```

```
'makes inner function'
```

**`__name__`** имя функции, изменяемое

```
>>> make_function.__name__
```

```
'make_function'
```

**`__qualname__`** fully qualified имя, изменяемое

```
>>> make_function.__qualname__
```

```
'make_function'
```

```
>>> fn.__qualname__
```

```
'make_function.<locals>.inner'
```

## ФУНКЦИИ: атрибуты

**`__defaults__`** кортеж дефолтных значений, изменяемое

```
>>> fn.__defaults__
```

```
(999,)
```

**`__kwdefaults__`** словарь дефолтных значений квартгов, изменяемое

```
>>> make_function.__kwdefaults__
```

```
{'kw': 12}
```

**`__closure__`** кортеж свободных переменных функции

```
>>> make_function.__closure__
```

```
None
```

```
>>> fn.__closure__[0].cell_contents
```

```
(54,)
```

# Пространство имён

*“Namespaces are one honking great idea --  
let's do more of those!”*

Tim Peters (`import this`)

# Пространство имён

Пространство имён — это совокупность определенных в настоящий момент символьических имен и информации об объектах, на которые они ссылаются.

- Встроенное
- Глобальное
- Объемлющее
- Локальное

# Пространство имён

Область видимости имени это часть программы, в которой данное имя обладает значением.

Интерпретатор определяет эту область в среде выполнения, основываясь на том, где располагается определение имени и из какого места в коде на него ссылаются.

Область видимости: LEGB

1. Локальная
2. Объемлющая
3. Глобальная
4. Встроенная

- `globals()`
- `locals()`
- `global`
- `nonlocal`

## \_\_builtins\_\_

```
>>> hasattr(__builtins__, "dir")
```

```
True
```

```
>>> dir(__builtins__)
```

```
...
```

### \_\_builtins\_\_

```
int float str bool tuple list dict set map zip filter range enumerate
sorted min max reversed len sum all any globals locals callable dir
type isinstance issubclass hasattr getattr setattr delattr
```

# Встроенные функции

# map

```
map(function, iterable, [iterable 2, iterable 3, ...])
```

```
def func(el1, el2):
```

```
    return '%s|%s' % (el1, el2)
```

```
list(map(func, [1, 2], [3, 4, 5])) # ['1|3', '2|4']
```

Применяет указанную функцию к каждому элементу указанной последовательности/последовательностей.

# reduce

```
from functools import reduce  
reduce(function, iterable[, initializer])  
  
items = [1,2,3,4,5]  
  
sum_all = reduce(lambda x,y: x + y, items)
```

Применяет указанную функцию к элементам последовательности, сводя её к единственному значению.

# filter

```
filter(function, iterable)

def is_even(x):

    return x % 2 == 0:

>>> print(list(filter(is_even, [1, 3, 2, 5, 20, 21])))

[2, 20]
```

Функция filter предлагает элегантный вариант фильтрации элементов последовательности. Принимает в качестве аргументов функцию и последовательность, которую необходимо отфильтровать.

# zip

```
>>> a = [1, 2, 3]
>>> b = "xyz"
>>> c = (None, True)
>>> print(list(zip(a, b, c)))
[(1, 'x', None), (2, 'y', True)]
```

Функция zip объединяет в кортежи элементы из последовательностей переданных в качестве аргументов.

# compile

```
compile(source, filename, mode, flag, dont_inherit, optimize)

# выполнение в exec
>>> x = compile('x = 1\nz = x + 5\nprint(z)', 'test', 'exec')
>>> exec(x)
# 6

# выполнение в eval
>>> y = compile("print('4 + 5 =', 4+5)", 'test', 'eval')
>>> eval(y)
# 4 + 5 = 9
```

## exec

```
exec(obj[, globals[, locals]]) -> None
```

Динамически исполняет указанный код.

# eval

`eval(expression[, globals[, locals]])`

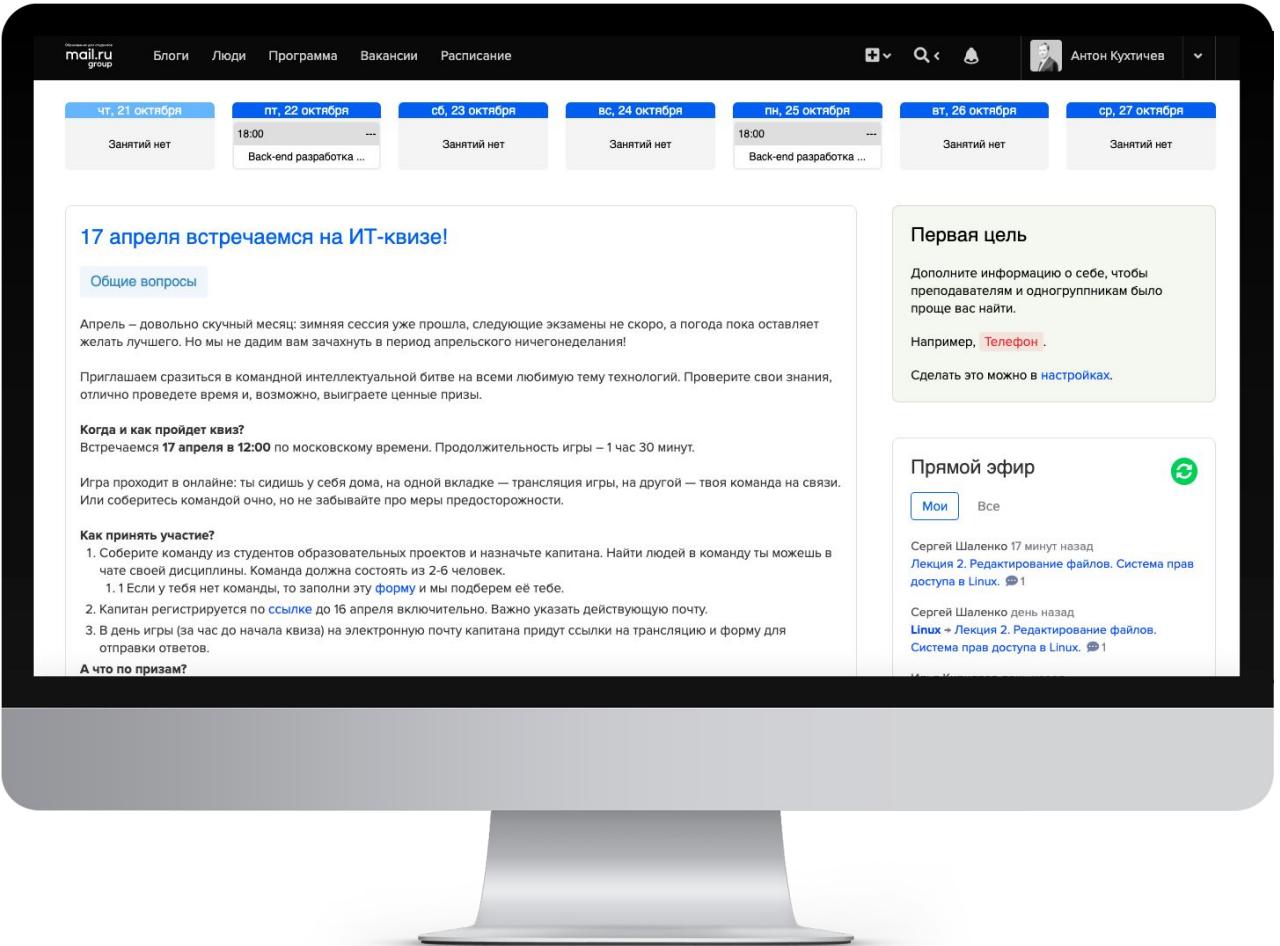
- в eval() запрещены операции присваивания;
- SyntaxError также вызывается в случаях, когда eval() не удается распарсить выражение из-за ошибки в записи;
- Аргумент `globals` опционален. Он содержит словарь, обеспечивающий доступ eval() к глобальному пространству имен;
- В `locals`-словарь содержит переменные, которые eval() использует в качестве локальных имен при оценке выражения.

# Домашнее задание

- Написать функцию, которая в качестве аргументов принимает:
  - строку json;
  - список необходимых полей;
  - список ключевых слов, которые будут искааться в необходимых полях;
  - функция-обработчик слов, которые встретились в списке ключевых слов
- Использовать mock-объект при тестировании;
- Использовать factory boy;
- Узнать степень покрытия тестами с помощью библиотеки coverage

# Напоминание оставить отзыв

Это правда важно



Спасибо за  
внимание!

Вопросы?

# Углубленный Python

Лекция 3

Объектная модель, введение в ООП

Кандауров Геннадий



образование

# Напоминание отметиться на портале

+ ОСТАВИТЬ ОТЗЫВ

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

Below the schedule, there's a blog post titled 'Углубленный Python' (Advanced Python) with the subtitle 'Blog for materials on the "Advanced Python" course'. It has 57 readers and 2 topics. A search bar and a 'Найти' (Find) button are below the post. To the right, there's a sidebar for 'Прямой эфир' (Live Stream) and a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

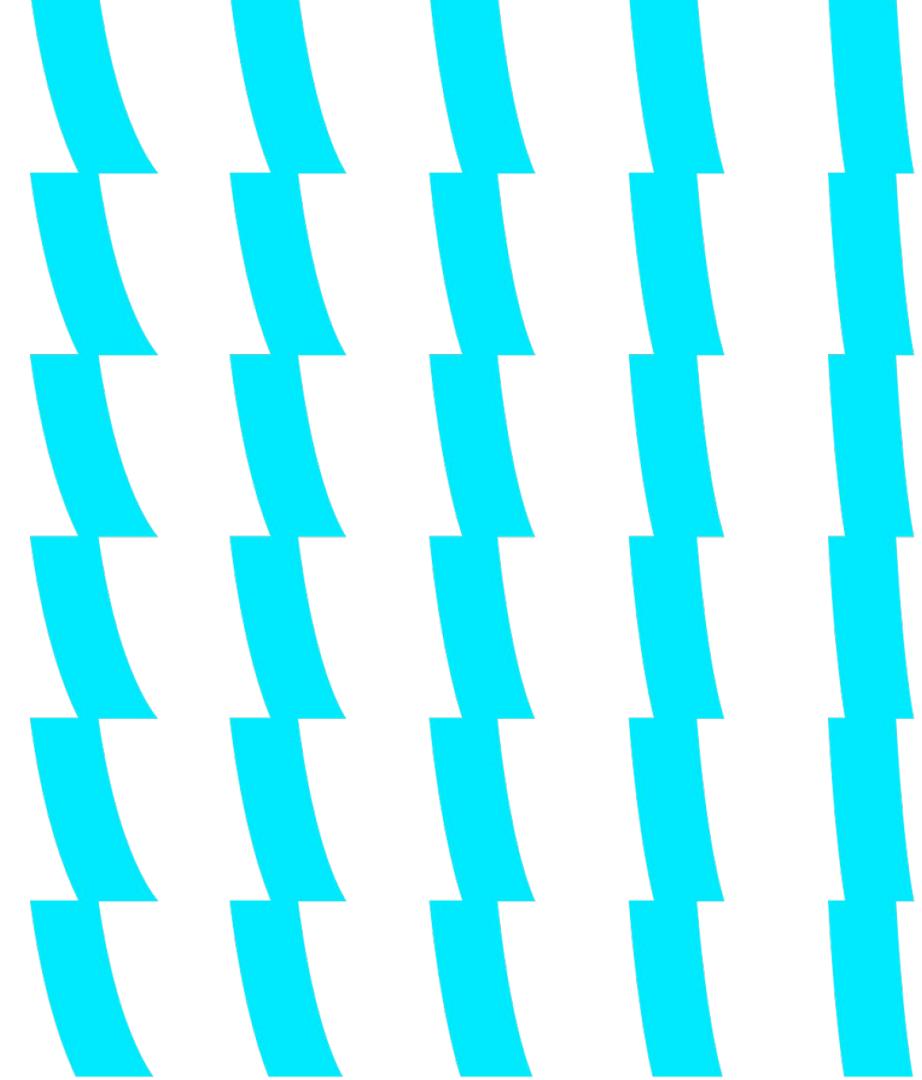
Квиз про прошлой лекции

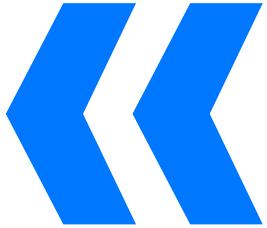


# Содержание занятия

1. Классы
2. ООП

# Классы





*Objects are Python's abstraction for data. All data in a Python program is represented by objects or by relations between objects.*

[docs.python.org](https://docs.python.org)

## Классы: атрибуты

```
class A:  
    name = "cls_name"  
    __cls_private = "cls_private"  
  
    def __init__(self, val):  
        self.val = val  
        self._protected = "protected"  
        self.__private = "private"  
  
    def print(self):  
        print(  
            f"{self.val=}, {self._protected=}, {self.__private=}, "  
            f"{self.name=}, {self.__cls_private=}")  
    )
```

## Классы: свойства

```
# классический подход

class Author:

    def __init__(self, name):
        self.__name = ""
        self.set_name(name)

    def get_name(self):
        return self.__name

    def set_name(self, val):
        self.__name = val
```

```
# pythonic

class Author:

    def __init__(self, name):
        self.name = name

    @property
    def name(self):
        return self.__name

    @name.setter
    def name(self, val):
        self.__name = val
```

# Классы: свойства

```
class Author:  
    def __init__(self, name):  
        self.name = name  
  
    @property  
    def name(self):  
        """name doc"""  
        return self.__name  
  
    @name.setter  
    def name(self, val):  
        self.__name = val  
  
    @name.deleter  
    def name(self, val):  
        self.__name = val
```

```
class Author:  
    def __init__(self, name):  
        self.name = name  
  
    def get_name(self):  
        return self.__name  
  
    def set_name(self, val):  
        self.__name = val  
  
    def del_name(self):  
        del self.__name  
  
    name = property(  
        get_name,  
        set_name,  
        del_name,  
        "name doc",  
    )
```

# Классы: свойства read/write only

```
class Author:  
    def __init__(self, name, password):  
        self.__name = name  
        self.password_hash = None  
        self.password = password  
  
    @property  
    def name(self):  
        """name is read-only"""  
        return self.__name  
  
    @property  
    def password(self):  
        raise AttributeError("Password is write-only")  
  
    @password.setter  
    def password(self, plaintext):  
        self.password_hash = make_hash_from_password(plaintext)
```

# Классы: методы

```
class A:  
    @staticmethod  
    def print_static():  
        print("static")  
  
    @classmethod  
    def print_cls(cls):  
        print(f"class_method for {cls.__name__}")  
  
    def __init__(self, val):  
        self.val = val  
  
    def print_offset(self, offset=10):  
        print(self.val + offset)  
  
    def __str__(self):  
        return f"{self.__class__.__name__}:val={self.val}"
```

# Классы: доступ к атрибутам

Чтобы найти атрибут объекта `obj`, python обыскивает:

1. Сам объект (`obj.__dict__` и его системные атрибуты)
2. Класс объекта (`obj.__class__.__dict__`).
3. Классы, от которых унаследован класс объекта (`obj.__class__.__mro__`)

# Классы: MRO

**Порядок разрешения методов** (method resolution order) позволяет python выяснить, из какого класса-предка нужно вызывать метод, если он не обнаружен непосредственно в классе-потомке.

```
cls.__mro__
```

```
cls.mro()
```

```
>>> B.mro()
```

```
[__main__.B, __main__.A, object]
```

# Классы: локальный порядок старшинства

```
>>> class A:  
...     pass  
...  
>>> class B:  
...     pass  
...  
>>> class C(A, B):  
...     pass  
...  
>>> C.mro()  
[<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'object'>]  
>>>  
>>> class C(B, A):  
...     pass  
...  
>>> C.mro()  
[<class '__main__.C'>, <class '__main__.B'>, <class '__main__.A'>, <class 'object'>]
```

```
graph TD; object --> A; object --> B; A --> C; B --> C;
```

# Классы: магические атрибуты

## Классы

`__name__` — имя класса

`__module__` — модуль, в котором объявлен класс

`__qualname__` — fully qualified имя

`__doc__` — докстринг

`__annotations__` — аннотации статических полей класса

`__dict__` — namespace класса

## Методы

`__self__` — объект класса

`__func__` — сама функция, которую мы в классе объявили

# Классы: магические атрибуты

## Поля, относящиеся к наследованию

`__bases__` — базовые классы

`__base__` — базовый класс, который указан первым по порядку

`__mro__` — список классов, упорядоченный по вызову функции `super`

```
class B(A): pass
```

```
>>> B.__bases__
```

```
(__main__.A,)
```

```
>>> B.__base__
```

```
__main__.A
```

```
>>> B.__mro__
```

```
(__main__.B, __main__.A, object)
```

# Классы: магические методы

`object.__new__(cls[, ...])`

Статический метод, создает новый экземпляр класса.

После создания экземпляра вызывается (уже у экземпляра) метод `__init__`.  
`__init__` ничего не должен возвращать (кроме `None`), иначе - `TypeError`

```
class Singleton:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super().__new__(cls, *args, **kwargs)
        return cls._instance
```

# Классы: магические методы

## Доступ к атрибутам

- `__getattribute__(self, name)`
- `__getattr__(self, name)`
- `__setattr__(self, name, val)`
- `__delattr__(self, name)`
- `__dir__(self)`

## Классы: магические методы

```
object.__call__(self[, args...])
```

```
class Adder:  
    def __init__(self, val):  
        self.val = val  
  
    def __call__(self, value):  
        return self.val + value  
  
a = Adder(10)  
a(5) # 15
```

# Классы: магические методы

## To string

`__repr__` — представление объекта. Если возможно, должно быть валидное python выражение для создания такого же объекта

`__str__` — вызывается функциями `str`, `format`, `print`

`__format__` — вызывается при форматировании строки

# Классы: магические методы

## Сравнение

`object.__lt__(self, other)`

`object.__le__(self, other)`

`object.__eq__(self, other)`

`object.__ne__(self, other)`

`object.__gt__(self, other)`

`object.__ge__(self, other)`

`x < y == x.__lt__(y) # <=, ==, !=, >, >=`

# Классы: магические методы

## Эмуляция чисел

```
object.__add__(self, other)
object.__sub__(self, other)
object.__mul__(self, other)
object.__matmul__(self, other) (@)
object.__truediv__(self, other)
object.__floordiv__(self, other)
object.__mod__(self, other)
object.__divmod__(self, other)
object.__pow__(self, other[, modulo])
object.__lshift__(self, other)
object.__rshift__(self, other)
object.__and__(self, other)
object.__xor__(self, other)
object.__or__(self, other)
```

# Классы: магические методы

## Эмуляция чисел

Методы вызываются, когда выполняются операции (+, -, \*, @, /, //, %, divmod(), pow(), \*\*, <<, >>, &, ^, |) над объектами

```
x + y == x.__add__(y)
```

Есть все такие же с префиксом r и i:

`__radd__` - вызывается, если левый операнд не поддерживает `__add__`

`__iadd__` - вызывается, когда `x += y`

# Классы: магические методы

## Эмуляция контейнеров

```
object.__len__(self)
object.__length_hint__(self)
object.__getitem__(self, key)
object.__setitem__(self, key, value)
object.__delitem__(self, key)
object.__missing__(self, key)
object.__iter__(self)
object.__next__(self)
object.__reversed__(self)
object.__contains__(self, item)
```

# Классы: магические методы

## `__hash__`

Вызывается функцией `hash()` и коллекциями, которые построены на основе hash-таблиц. Нужно, чтобы у равных объектов был одинаковый hash.

Если определен метод `__eq__` и не определен `__hash__`, то объект не может быть ключом в `hashable` коллекции.

```
>>> key1 = (1, 2, 3)
>>> key2 = (1, 2, 3, [4, 5])
>>> s = set()
>>> s.add(key1) # ???
>>> s.add(key2) # ???
```

# Классы: магические методы

## `__slots__`

Позволяет явно указать поля, которые будут в классе.

В случае указания `__slots__` пропадают поля `__dict__` и `__weakref__`.

Используя `__slots__` можно экономить на памяти и времени доступа к атрибутам объекта.

```
class Point:  
    __slots__ = ('x', 'y')  
  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y
```

# Классы: наследование

```
class Timing:  
    def __init__(self, start, end):  
        self.start = start  
        self.end = end  
  
    def duration(self):  
        print("Timing.duration")  
        return self.end - self.start  
  
  
class MinuteTiming(Timing):  
    def duration(self):  
        print("MinuteTiming.duration")  
        seconds = super().duration()  
        return seconds / 60  
  
  
        >>> m = MinuteTiming(1000, 7000)  
        >>> m.duration()  
        MinuteTiming.duration  
        Timing.duration  
        100.0
```

## Классы: \_\_init\_subclass\_\_

```
class Timing:  
    def __init__(self, start, end):  
        self.start = start  
        self.end = end  
  
    @classmethod  
    def __init_subclass__(cls, **kwargs):  
        print("INIT subclass", cls, kwargs)  
  
class MinuteTiming(Timing):  
    def duration(self):  
        print("MinuteTiming.duration")  
        seconds = super().duration()  
        return seconds / 60
```

## Домашнее задание #03

- Реализовать кастомный список,  
унаследованный от `list`
- +тесты
- `flake8 + pylint` перед сдачей

# Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

Below the schedule, a blog post titled 'Углубленный Python' is shown. It has 57 readers and 2 topics. There are buttons for 'Подписаться' (Subscribe) and 'Создать топик' (Create topic). A search bar and a 'Найти' (Find) button are also present.

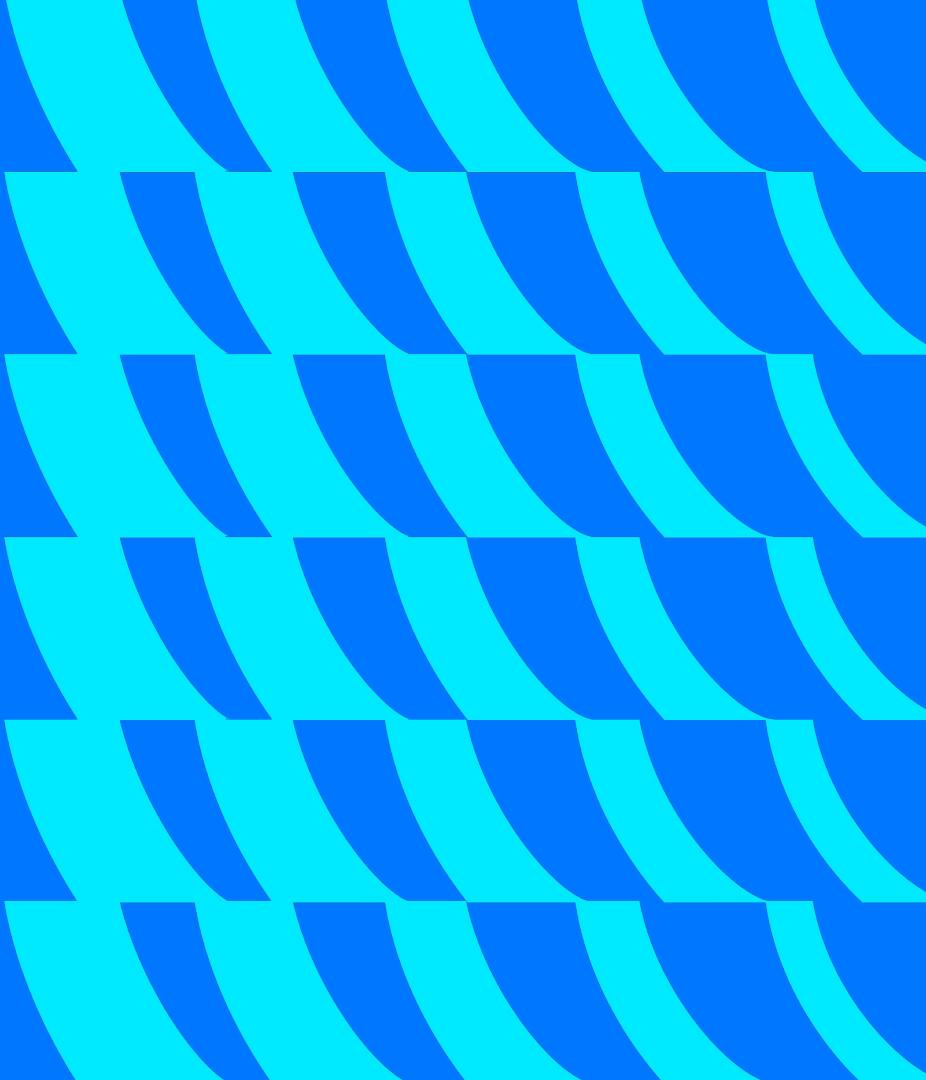
On the right side, there's a sidebar for 'Прямой эфир' (Live Stream) with a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

A modal window at the bottom left says 'Добро пожаловать на курс!' (Welcome to the course!) with 'Изменить' (Change) and 'Удалить' (Delete) buttons. A note below it says 'Всем привет и логином пожаловать на курс по углубленному изучению Python!'

Спасибо за  
внимание



образование



# Углубленный Python

Лекция 4

Метапрограммирование,  
дескрипторы, АВС

Кандауров Геннадий



образование

# Напоминание отметиться на портале

+ ОСТАВИТЬ ОТЗЫВ

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

Below the schedule, there's a blog post titled 'Углубленный Python' (Advanced Python) with the subtitle 'Blog for materials on the "Advanced Python" course'. It has 57 readers and 2 topics. A search bar and a 'Найти' (Find) button are below the post. To the right, there's a sidebar for 'Прямой эфир' (Live Stream) and a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

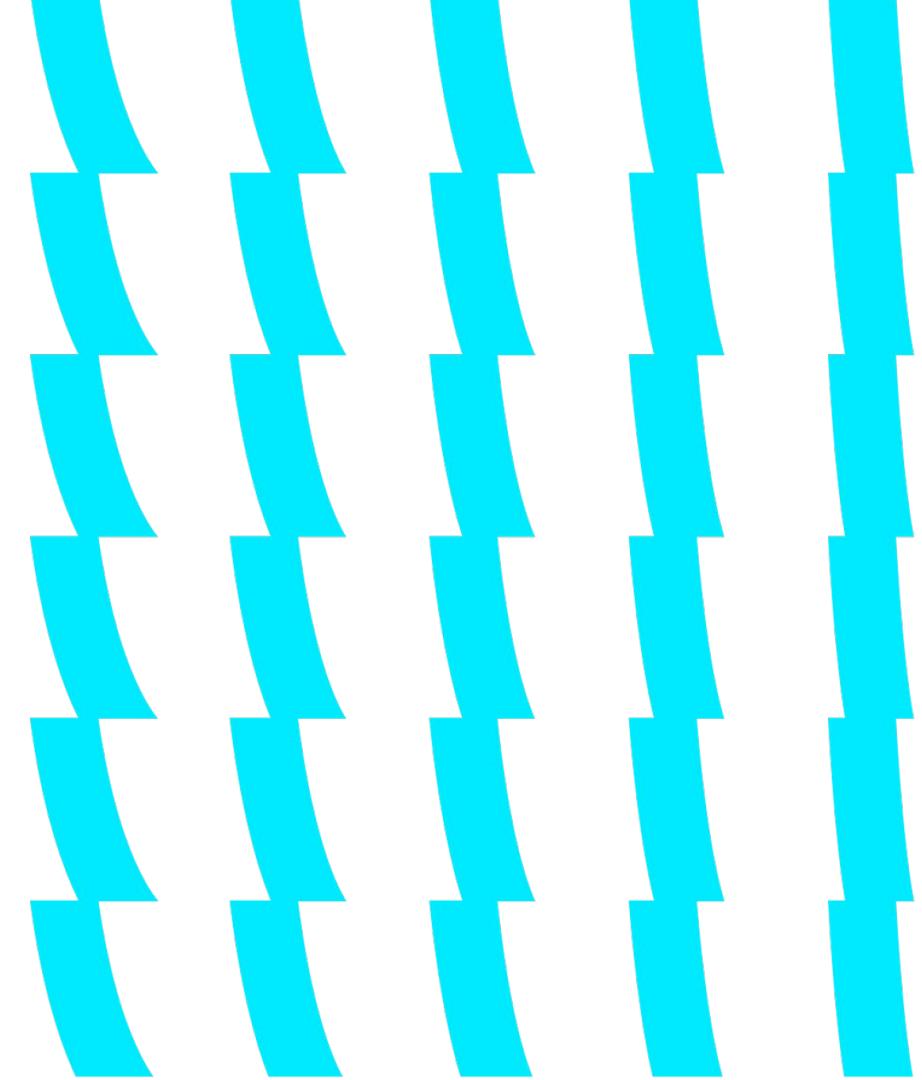
Квиз про прошлой лекции



# Содержание занятия

1. Классы
2. Дескрипторы
3. Метапрограммирование
4. ABC

# Классы



# Классы

```
class A:  
    @staticmethod  
    def print_static():  
        print("static")  
  
    @classmethod  
    def print_cls(cls):  
        print(f"class_method for {cls.__name__}")  
  
    def __init__(self, val):  
        self.val = val  
  
    def print_offset(self, offset=10):  
        print(self.val + offset)  
  
    def __str__(self):  
        return f"{self.__class__.__name__}:val={self.val}"
```

# Классы: доступ к атрибутам

Чтобы найти атрибут объекта `obj`, python обыскивает:

1. Сам объект (`obj.__dict__` и его системные атрибуты)
2. Класс объекта (`obj.__class__.__dict__`).
3. Классы, от которых унаследован класс объекта (`obj.__class__.__mro__`)

# Классы: магические методы

`object.__new__(cls[, ...])`

Статический метод, создает новый экземпляр класса.

После создания экземпляра вызывается (уже у экземпляра) метод `__init__`.  
`__init__` ничего не должен возвращать (кроме `None`), иначе - `TypeError`

```
class Singleton:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super().__new__(cls, *args, **kwargs)
        return cls._instance
```

# Классы: магические методы

`object.__del__(self)`

Финализатор

```
class Connector:  
    def __init__(self, db_name):  
        self.conn = DbDriver(db_name)  
    def __del__(self):  
        self.conn.close()  
        print("DEL")
```

```
db = Connector("users")  
del db # ???
```

## Классы: магические методы

`object.__set_name__(self, owner, name)`

Автоматически вызывается при создании класса-владельца `owner`.

Хук вызывается, когда объекту было присвоено имя `name` в этом классе

```
class Attr:  
    def __set_name__(self, owner, name):  
        print(f"locals()={locals()}")  
        self.name = name  
  
class A:  
    x = Attr() # Automatically calls: x.__set_name__(A, "x")  
  
a = A()
```

## Классы: магические методы

`object.__call__(self[, args...])`

```
class Adder:  
    def __init__(self, val):  
        self.val = val  
  
    def __call__(self, value):  
        return self.val + value  
  
a = Adder(10)  
a(5) # 15
```

# Классы: магические методы

## `object.__slots__`

Позволяет явно указать поля, которые будут в классе.

В случае указания `__slots__` пропадают поля `__dict__` и `__weakref__`.

Используя `__slots__` можно экономить на памяти и времени доступа к атрибутам объекта.

```
class Point:  
    __slots__ = ("x", "y")  
  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y
```

# Классы: магические методы

## Доступ к атрибутам

- `__getattribute__(self, name)`
- `__getattr__(self, name)`
- `__setattr__(self, name, val)`
- `__delattr__(self, name)`
- `__dir__(self)`

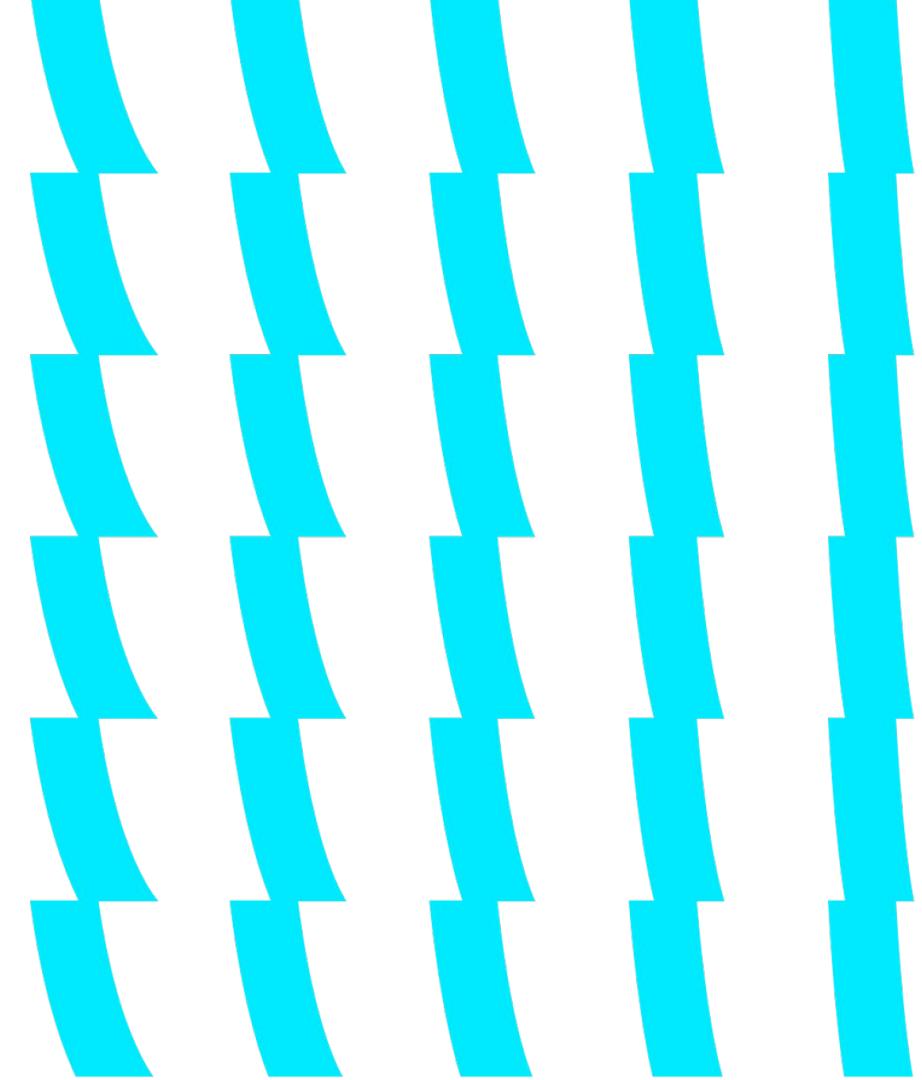
# Классы: наследование

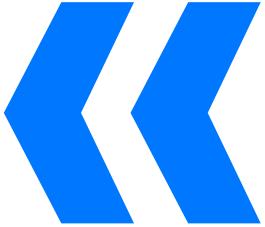
```
class Timing:  
    def __init__(self, start, end):  
        self.start = start  
        self.end = end  
  
    def duration(self):  
        print("Timing.duration")  
        return self.end - self.start  
  
  
class MinuteTiming(Timing):  
    def duration(self):  
        print("MinuteTiming.duration")  
        seconds = super().duration()  
        return seconds / 60  
  
  
        >>> m = MinuteTiming(1000, 7000)  
        >>> m.duration()  
        MinuteTiming.duration  
        Timing.duration  
        100.0
```

## Классы: \_\_init\_subclass\_\_

```
class Timing:  
    def __init__(self, start, end):  
        self.start = start  
        self.end = end  
  
    @classmethod  
    def __init_subclass__(cls, **kwargs):  
        print("INIT subclass", cls, kwargs)  
        super().__init_subclass__(**kwargs)  
  
  
class MinuteTiming(Timing):  
    def duration(self):  
        print("MinuteTiming.duration")  
        seconds = super().duration()  
        return seconds / 60
```

# **Дескрипторы**





*Дескриптор это атрибут объекта со “связанным поведением”, то есть такой атрибут, при доступе к которому его поведение переопределяется методом протокола дескриптора. Эти методы `__get__`, `__set__` и `__delete__`. Если хотя бы один из этих методов определен в объекте , то можно сказать что этот объект дескриптор.*

Раймонд Хеттингер

## Дескрипторы

- Если определен один из методов `__get__`, `__set__` и `__delete__`, объект считается дескриптором.
- Если объект дескриптора определяет `__get__`, `__set__`, то он считается data дескриптором.
- Если объект дескриптора определяет `__get__`, то является non-data дескриптором.

# Дескрипторы

```
>>> class A:  
    def foo(self):  
        pass  
    a = A()  
  
>>> a.foo.__class__.__get__  
<slot wrapper '__get__' of 'method' objects>  
  
>>> A.__dict__['foo'] # Внутренне хранится как функция  
<function foo at 0x00C45070>  
  
>>> A.foo # Доступ через класс возвращает несвязанный метод  
<unbound method A.foo>  
  
>>> a.foo # Доступ через экземпляр объекта возвращает связанный метод  
<bound method A.foo of <__main__.A object at 0x00B18C90>>
```

# Дескрипторы

```
class MyDescriptor:  
    def __get__(self, obj, objtype):  
        print(f"get {obj} cls={objtype}")  
  
    def __set__(self, obj, val):  
        print(f"set {val} for {obj}")  
  
    def __delete__(self, obj):  
        print(f"delete from {obj}")  
  
class MyClass:  
    field = MyDescriptor()
```

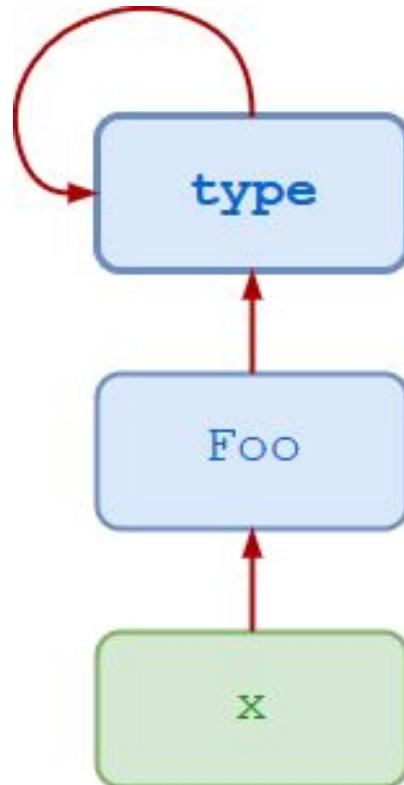
```
>>> inst = MyClass()  
>>> MyClass.field  
get None cls=<class '__main__.MyClass'>  
>>> inst.field  
get <__main__.MyClass object ...> cls=<class  
'__main__.MyClass'>  
>>> inst.field = 1  
set 1 for <__main__.MyClass object ...>  
>>> del inst.field  
delete from <__main__.MyClass object ...>
```

# Метаклассы

Классы, экземпляры которых  
являются классами

## Метаклассы: type

```
class Foo:  
    pass  
  
x = Foo()
```



## Метаклассы: type

Новые классы создаются с помощью вызова

```
type(<name>, <bases>, <classdict>)
```

name – имя класса (`__name__`)

bases – базовые классы (`__bases__`)

classdict – namespace класса (`__dict__`)

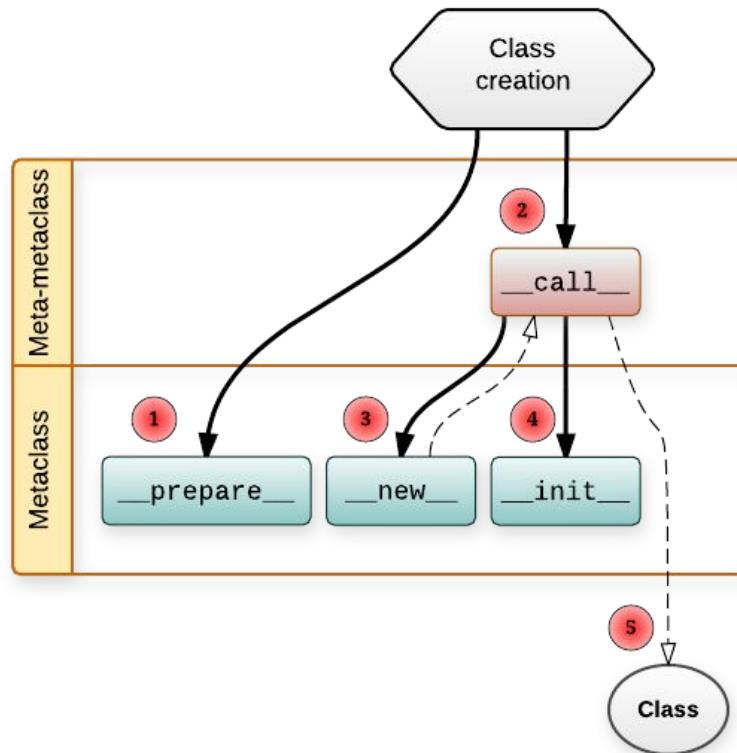
```
MyClass = type("MyClass", (), {})
```

## Метаклассы: type

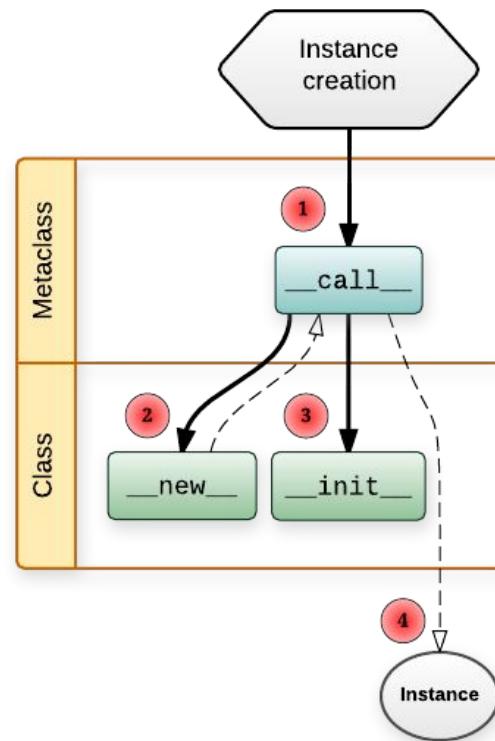
```
>>> Bar = type('Bar', (Foo,), dict(attr=100))
>>> x = Bar()
>>> x.attr
100
>>> x.__class__
<class '__main__.Bar'>
>>> x.__class__.__bases__
(<class '__main__.Foo'>,)

>>> class Bar(Foo):
...     attr = 100
...
>>> x = Bar()
>>> x.attr
100
>>> x.__class__.__bases__
(<class '__main__.Foo'>,)
```

# Метаклассы: создание класса



# Метаклассы



## Метаклассы: создание класса

- определяются базовые классы
- определяется метакласс
- подготавливается namespace класса (`__prepare__`)
- выполняется тело класса
- создается класс (`__new__`, `__init__`)

# Метаклассы

```
class AMeta(type):
    def __new__(mcs, name, bases, classdict, **kwargs):
        cls = super().__new__(mcs, name, bases, classdict)
        print('Meta __new__', cls)
        return cls

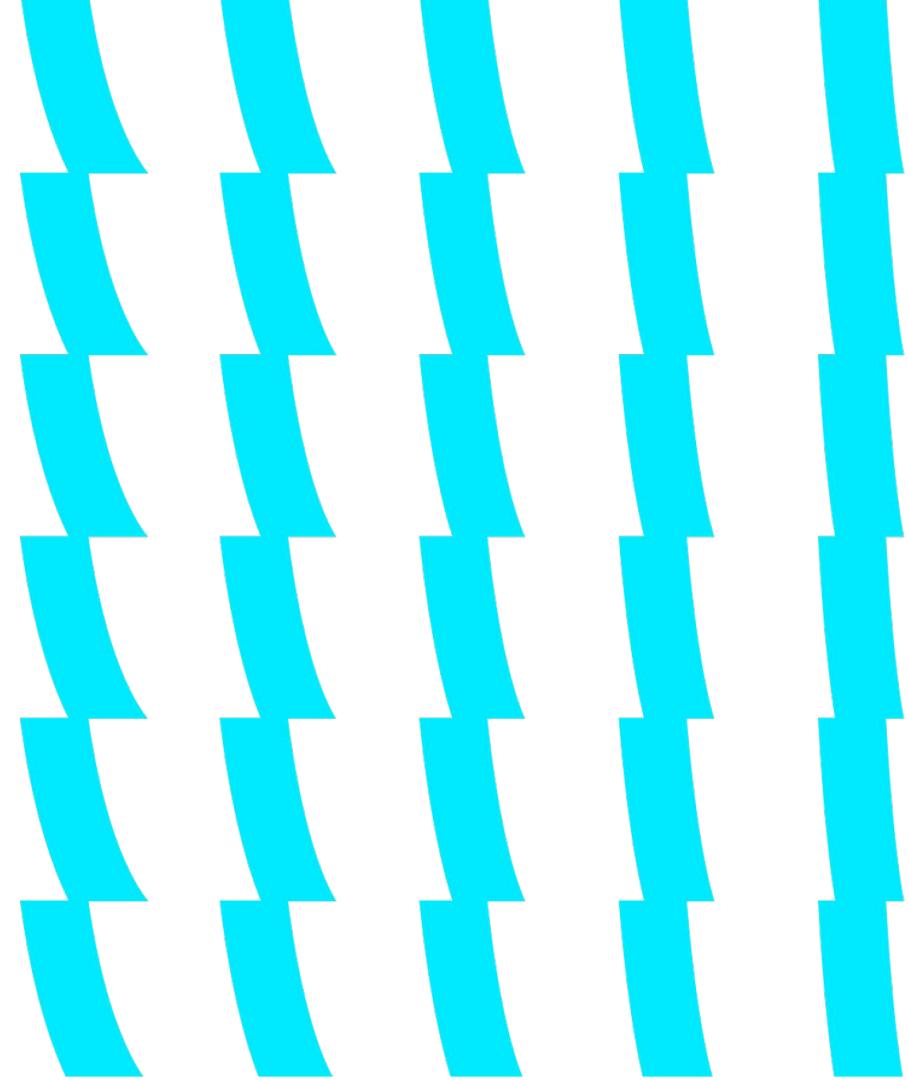
    def __init__(cls, name, bases, classdict, **kwargs):
        super().__init__(name, bases, classdict, **kwargs)

    def __call__(cls, *args, **kwargs):
        return super().__call__(*args, **kwargs)

    @classmethod
    def __prepare__(mcs, name, bases, **kwargs):
        print('Meta __prepare__', **kwargs)
        return {'b': 2, 'a': 2}
```

# ABC

Добавляем абстракции



# ABC

```
>>> from abc import ABCMeta, abstractmethod
>>> class C(metaclass=ABCMeta):
...     @abstractmethod
...     def abs_method(self):
...         pass
>>> c = C()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class C with abstract methods abs_method

>>> class B(C):
...     def abs_method(self):
...         print("Now a concrete method")
>>> b = B()
>>> b.abs_method()
Now a concrete method
```

# ABC

```
class Hashable(metaclass=ABCMeta):
    __slots__ = ()
    @abstractmethod
    def __hash__(self):
        return 0

    @classmethod
    def __subclasshook__(cls, C):
        if cls is Hashable:
            return _check_methods(C, "__hash__")
        return NotImplemented

>>> from collections.abc import Hashable
>>> isinstance("123", Hashable) # ???
>>> isinstance({}, Hashable) # ???
```

## Домашнее задание #4

- Реализация метакласса с префиксом `custom_`
- Дескрипторы с проверками типов и значений данных
- Тесты
- `flake8` и `pylint` перед сдачей

# Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

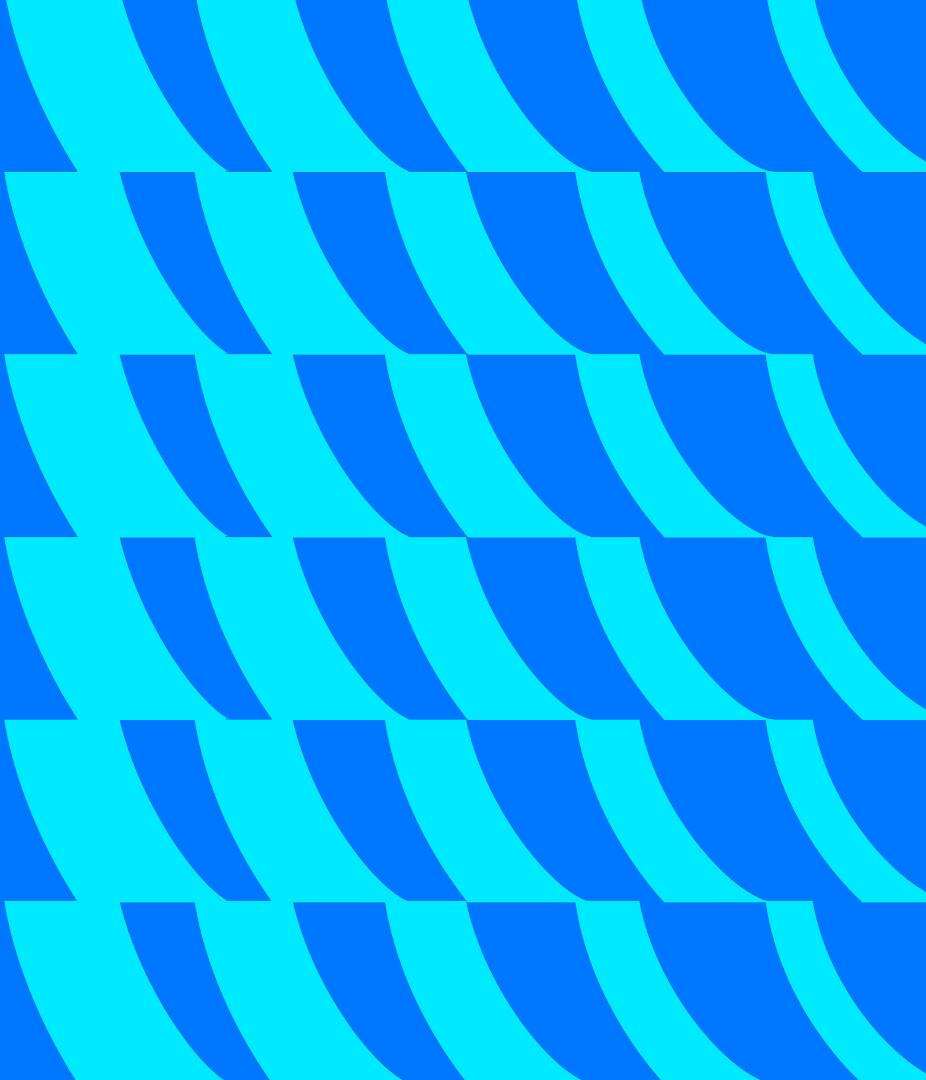
Below the schedule, a blog post titled 'Углубленный Python' is shown. It has 57 readers and 2 topics. There are buttons for 'Подписаться' (Subscribe) and 'Создать топик' (Create topic). A search bar and a 'Найти' (Find) button are also present.

On the right side, there's a sidebar for 'Прямой эфир' (Live Stream) with a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

Спасибо за  
внимание



образование



# Углубленный Python

## Лекция 5

### Стандартная библиотека

Кандауров Геннадий



образование

# Напоминание отметиться на портале

+ ОСТАВИТЬ ОТЗЫВ

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

In the center, there's a blog post titled 'Углубленный Python' (Advanced Python) with a description: 'Blog for materials on the "Advanced Python" course'. It has 57 readers and 2 topics. Below the post is a search bar and a 'Найти' (Find) button. To the right, there's a sidebar for 'Прямой эфир' (Live Stream) and a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

## Квиз про прошлой лекции

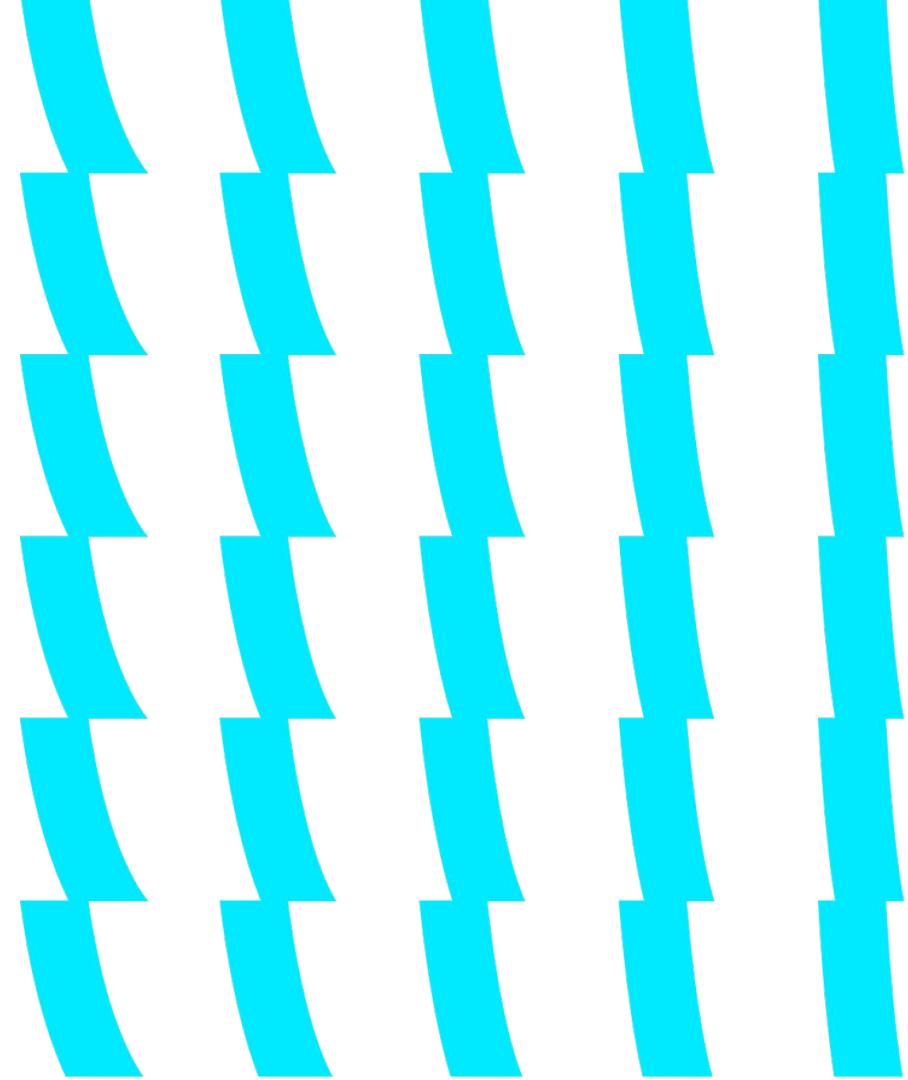


# Содержание занятия

1. ABC
2. Числа, строки
3. *collections*
4. *functools*
5. *itertools*
6. *heapq*
7. Файлы и каталоги

# ABC

Добавляем абстракции



# ABC

```
>>> from abc import ABCMeta, abstractmethod
>>> class C(metaclass=ABCMeta):
...     @abstractmethod
...     def abs_method(self):
...         pass
>>> c = C()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class C with abstract methods abs_method

>>> class B(C):
...     def abs_method(self):
...         print("Now a concrete method")
>>> b = B()
>>> b.abs_method()
Now a concrete method
```

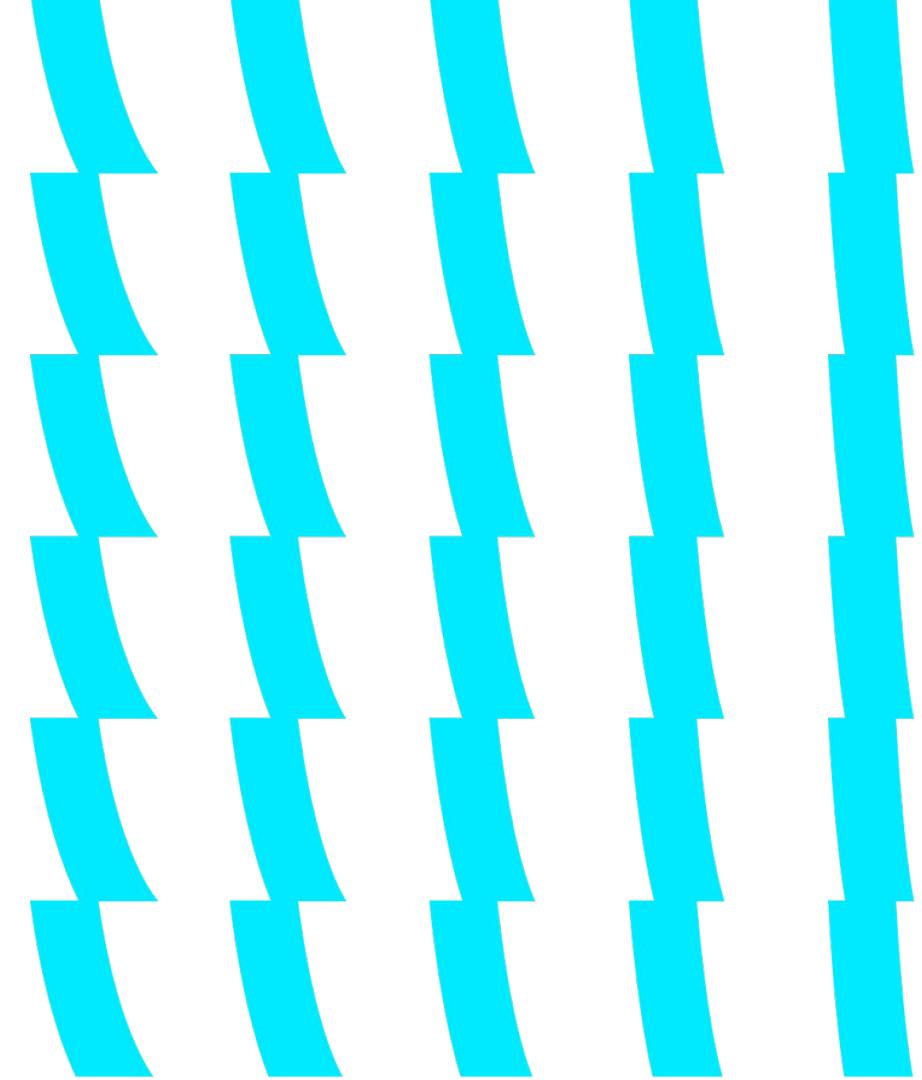
# ABC

```
class Hashable(metaclass=ABCMeta):
    __slots__ = ()
    @abstractmethod
    def __hash__(self):
        return 0

    @classmethod
    def __subclasshook__(cls, C):
        if cls is Hashable:
            return _check_methods(C, "__hash__")
        return NotImplemented

>>> from collections.abc import Hashable
>>> isinstance("123", Hashable) # ???
>>> isinstance({}, Hashable) # ???
```

# Числа



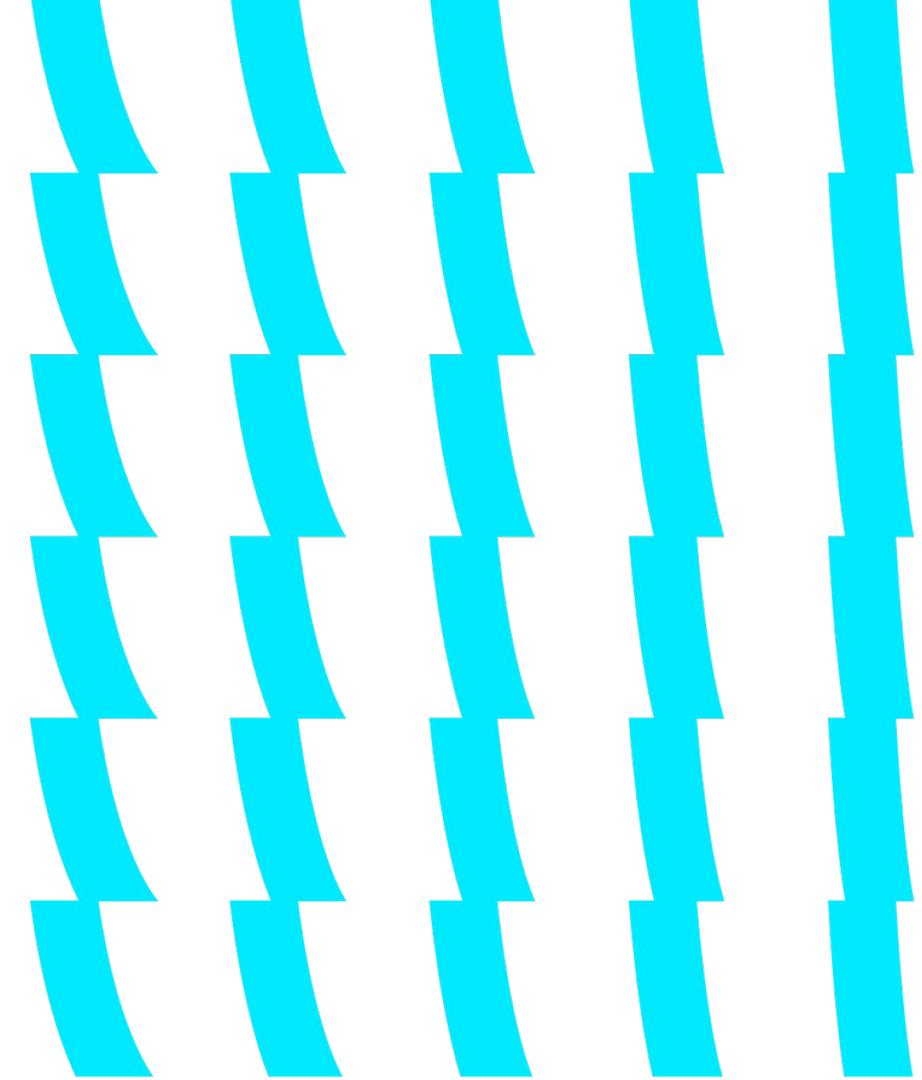
# float

```
>>> float("-inf"), float("inf"), float("nan")
(-inf, inf, nan)
>>> 0.1 + 0.2 == 0.3
False
>>> 0.1 + 0.2 <= 0.3
False
>>> 0.1 + 0.2
0.3000000000000004
>>> (0.1).as_integer_ratio()
(3602879701896397, 36028797018963968)
>>> format(0.1, ".25f")
'0.100000000000000055511151'
>>> math.isclose(0.1 + 0.2, 0.3)
True
```

# Decimal, Fraction

```
>>> from decimal import Decimal  
>>> Decimal("0.1") + Decimal("0.2") == Decimal("0.3")  
True  
>>> Decimal(1) / Decimal(7)  
Decimal('0.1428571428571428571428571429')  
  
>>> from fractions import Fraction  
>>> Fraction(1, 10)  
Fraction(1, 10)  
>>> Fraction(1, 10) + Fraction(2, 10) == Fraction(3, 10)  
True
```

# Строки



## str

- `isalpha()`
- `isascii()`
- `isidentifier()`
- `isalnum()`
- `isdecimal()`
- `isdigit()`
- `isnumeric()`

```
>>> s = "1²²³⁴⁴"  
>>> s.isalnum() # ???  
>>> s.isdigit() # ???  
>>> s.isnumeric() # ???  
>>> s.isdecimal() # ???
```

# str

- `startswith(prefix[, start[, end]])`
- `endswith(suffix[, start[, end]])`
- `capitalize()`
- `upper()`
- `isupper()`
- `lower()`
- `islower()`
- `title()`
- `istitle()`
- `swapcase()`
- `isprintable()`
- `isspace()`

## str

- `count(sub[, start[, end]])`
- `find(sub[, start[, end]]), rfind(sub[, start[, end]])`
- `index(sub[, start[, end]]), rindex(sub[, start[, end]])`
- `replace(old, new[, count])`
- `center(width[, fillchar])`
- `encode(encoding='utf-8', errors='strict')`
- `expandtabs(tabsize=8)`
- `format(*args, **kwargs)`
- `ljust(width[, fillchar]), rjust(width[, fillchar])`
- `lstrip([chars]), strip([chars]), rstrip([chars])`

## str

- `split(sep=None, maxsplit=-1), rsplit(sep=None, maxsplit=-1)`
- `splitlines(keepends=False)`
- `partition(sep), rpartition(sep)`
- `join(iterable)`
- `zfill(width)`
- `removeprefix(prefix, /)`
- `removesuffix(suffix, /)`

# **collections**

Специализированные контейнерные  
типы данных, предоставляющие  
альтернативы для встроенных  
`dict`, `list`, `set` и `tuple`

## `collections.namedtuple`

```
namedtuple(typename, field_names, *, rename=False, defaults=None, module=None)
```

```
>>> Point = collections.namedtuple("Point", ["x", "y"])
```

```
>>> p = Point(11, y=22) # p = (11, 22)
```

```
>>> p[0] + p[1]
```

```
33
```

```
>>> x, y = p
```

```
>>> x, y
```

```
(11, 22)
```

```
>>> p.x + p.y
```

```
33
```

```
>>> p._asdict() # {'x': 1, 'y': 4}
```

## [collections.defaultdict](#)

```
collections.defaultdict([default_factory[, ...]])
```

Словарь, у которого по умолчанию всегда вызывается функция `default_factory`.

```
>>> import collections  
>>> defdict = collections.defaultdict(list)  
>>> print(defdict)  
defaultdict(<class 'list'>, {})  
>>> for i in range(5):  
...     defdict[i].append(i)  
>>> print(defdict)  
defaultdict(<class 'list'>, {0: [0], 1: [1], 2: [2], 3: [3], 4: [4]})
```

# [collections.OrderedDict](#)

`collections.OrderedDict([items])`

Словарь, который помнит порядок, в котором ему были даны ключи.

```
>>> import collections
>>> d = collections.OrderedDict(
...     [("a", "A"), ("b", "B"), ("c", "C")]
... )
>>> for k, v in d.items():
...     print(k, v)
'a', 'A'
'b', 'B'
'c', 'C'
>>> d.move_to_end("b")
```

## [collections.Counter](#)

`collections.Counter([iterable-or-mapping])`

Словарь для подсчёта хешируемых объектов.

- `elements()`
- `most_common([n])`
- `subtract([iterable-or-mapping])`
- `update([iterable-or-mapping])`

```
>>> words = re.findall(r"\w+", open("hamlet.txt").read().lower())
>>> Counter(words).most_common(5)
[('the', 1143), ('and', 966), ('to', 762), ('of', 669), ('i', 631)]
```

# `collections.deque`

`collections.deque([iterable[, maxlen]])`

Двусторонняя очередь из итерируемого объекта с максимальной длиной maxlen.

Добавление и удаление элементов в начало или конец выполняется за константное время.

- `append(x)`
- `appendleft(x)`
- `extend(iterable)`
- `extendleft(iterable)`
- `insert(i, x)`
- `pop()/popleft()`
- `remove(value)`

```
>>> d = deque("ghi")
>>> d.append("j")
>>> d.appendleft("f")
>>> d
deque(['f', 'g', 'h', 'i', 'j'])
>>> d.pop()
'j'
>>> d.popleft()
'f'
>>> d
deque(['g', 'h', 'i'])
```

# **functools**

Для функций высшего порядка

## functools

- `cache(user_function)`
- `cached_property(func)`
- `lru_cache(user_function)`
- `lru_cache(maxsize=128, typed=False)`

```
@functools.cache
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n - 1)
```

## singledispatch, singledispatchmethod(func)

```
@functools.singledispatch
def fun(arg, prefix="Hello"):
    print(f"{prefix} any type {arg}")

@fun.register(str)
def _(arg, prefix="Hi"):
    print(f"{prefix} str {arg}")

@fun.register
def _(arg: int, prefix="Hello"):
    print(f"{prefix} int {arg}")

>>> fun(123) # Hello int 123
>>> fun.registry.keys()
# dict_keys([<class 'object'>, <class 'str'>, <class 'int'>])
```

# functools

- `partial(func, /, *args, **keywords)`
- `partialmethod(func, /, *args, **keywords)`
- `total_ordering`
- `reduce(function, iterable[, initializer])`
- `wraps(wrapped, assigned=WRAPPER_ASSIGNMENTS, updated=WRAPPER_UPDATES)`
- `update_wrapper(wrapper, wrapped, assigned=WA, updated=WU)`

```
>>> basetwo = partial(int, base=2)
>>> basetwo("10010")
```

# **itertools**

Можно бесконечно смотреть на  
бесконечный цикл

# Итераторы

**Итератор** представляет собой объект-перечислитель, который для данного объекта выдает следующий элемент, либо вызывает исключение, если элементов больше нет.

```
num_list = [1, 2, 3]
itr = iter(num_list)
print(next(itr)) # 1
print(next(itr)) # 2
print(next(itr)) # 3
print(next(itr)) # StopIteration
```

```
class SpecialIterator:
    def __init__(self, limit):
        self.limit = limit
        self.counter = 0
    def __next__(self):
        if self.counter < self.limit:
            self.counter += 1
            return self.counter
        else:
            raise StopIteration
```

```
iter_obj = SpecialIterator(3)
print(next(iter_obj))
```

# ФУНКЦИИ: генераторы

Генератор – функция, которая при вызове `next()` возвращает следующий объект по алгоритму ее работы. Вместо `return` в генераторе используем `yield` (или вместе).

```
def gen(count):
    while count > 0:
        yield count
        count -= 1
    return count # будет аргументом StopIteration

for i in gen(5):
    print(i) # 5, 4, 3, 2, 1
```

## Itertools: бесконечные итераторы

- `count(start=0, step=1)`
- `cycle(iterable)`
- `repeat(object[, times])`

# Itertools

- `accumulate(iterable[, func, *, initial=None])`
- `chain(*iterables)`
- `compress(data, selectors)`
- `dropwhile(predicate, iterable)`
- `takewhile(predicate, iterable)`
- `filterfalse(predicate, iterable)`
- `groupby(iterable, key=None)`
- `islice(iterable, start, stop[, step])`
- `pairwise(iterable)`
- `starmap(function, iterable)`
- `tee(iterable, n=2)`
- `zip_longest(*iterables, fillvalue=None)`

## Itertools: комбинаторика

- `product(*iterables, repeat=1)`
- `permutations(iterable, r=None)`
- `combinations(iterable, r)`
- `combinations_with_replacement(iterable, r)`

# **Работа с файлами**

# Типы операций с файлами

- связанные с его открытием: открытие, закрытие файла, запись, чтение, перемещение по файлу и др.
- выполняющиеся без его открытия: работа с файлом как элементом файловой системы - переименование, копирование, получение атрибутов и др.

# Файловый объект

При открытии файла операционная система возвращает специальный дескриптор файла (идентификатор), однозначно определяющий, с каким файлом далее будут выполняться операции.

В Python работа с файлами осуществляется через специальный абстрактный файловый объект. В зависимости от способа создания такого объекта, он может быть привязан как к физическому файлу на диске, так и другому устройству, поддерживающему схожие операции (стандартный ввод/вывод и пр.).

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,  
newline=None, closefd=True, opener=None)
```

```
# кодировка  
import locale  
locale.getpreferredencoding(False)
```

# Файловый объект

```
f = open("some.file", "r")
data = f.read()
f.close()

# лучше
with open("some.file", "r") as f:
    data = f.read()
```

# Обработка файла

Режим	Описание
r	Только для чтения.
w	Только для записи. Создаст новый файл, если не найдет с указанным именем.
rb	Только для чтения (бинарный).
wb	Только для записи (бинарный). Создаст новый файл, если не найдет с указанным именем.
r+	Для чтения и записи.
rb+	Для чтения и записи (бинарный).
w+	Для чтения и записи. Создаст новый файл для записи, если не найдет с указанным именем.
wb+	Для чтения и записи (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
a	Откроет для добавления нового содержимого. Создаст новый файл для записи, если не найдет с указанным именем.
a+	Откроет для добавления нового содержимого. Создаст новый файл для чтения записи, если не найдет с указанным именем.
ab	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
ab+	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для чтения записи, если не найдет с указанным именем.

## Файловый объект: методы

- `file.close()`
- `file.fileno()`
- `file.flush()`
- `file.isatty()`
- `file.next()`
- `file.read(n)`
- `file.readline()`
- `file.readlines()`
- `file.seekable()`
- `file.tell()`
- `file.truncate(n)`
- `file.write(str)`
- `file.writelines(sequence)`

# Path

```
from pathlib import Path  
p = Path(".")  
pdf_path = p / "storage" / "slides.pdf"  
abs_path = "/usr" / p / "storage" / "slides.pdf"
```

```
p.is_dir(), p.is_file()
```

```
p.is_absolute()
```

```
p.exists()
```

```
p.glob(pat)
```

```
p.open(), p.read_text(), p.write_text()
```

```
p.unlink()
```

# **Разное**

# heapq

`heappush(heap, item)`

`heappop(heap)`

`heappushpop(heap, item)`

`heapify(x)`

`heapreplace(heap, item)`

`merge(*iterables, key=None, reverse=False)`

`nlargest(n, iterable, key=None)`

`nsmallest(n, iterable, key=None)`

## **Домашнее задание #5**

- LRU cache без OrderedDict
- Генератор по файлу с фильтром
- Тесты
- flake8 + pylint перед сдачей

# Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

Below the schedule, a blog post titled 'Углубленный Python' is shown. It has 57 readers and 2 topics. There are buttons for 'Подписаться' (Subscribe) and 'Создать топик' (Create topic). A search bar and a 'Найти' (Find) button are also present.

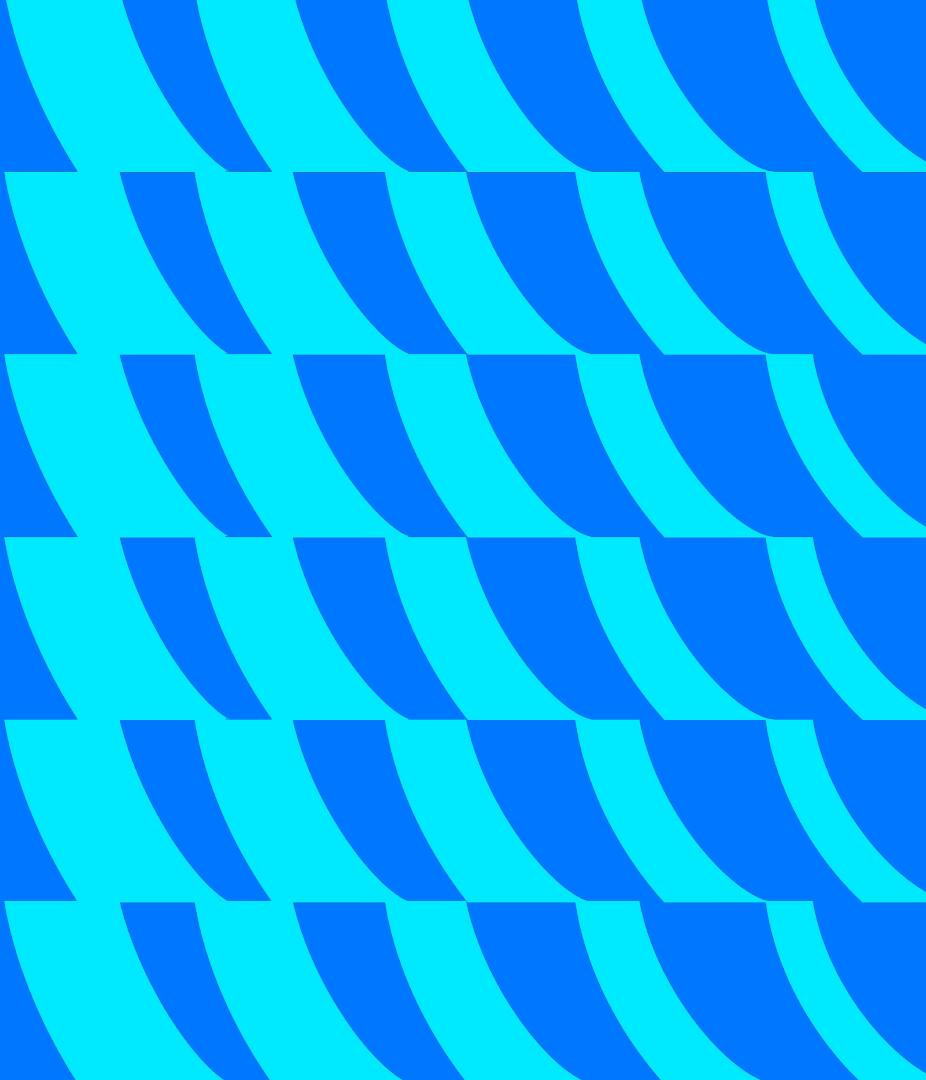
On the right side, there's a sidebar for 'Прямой эфир' (Live Stream) with a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

A modal window at the bottom left says 'Добро пожаловать на курс!' (Welcome to the course!) with 'Изменить' (Change) and 'Удалить' (Delete) buttons. A note below it says 'Всем привет и логином пожаловать на курс по углубленному изучению Python!'

Спасибо за  
внимание



образование



# Углубленный Python

Лекция 6

Потоки, GIL, процессы

Кандауров Геннадий



образование

# Напоминание отметиться на портале

+ ОСТАВИТЬ ОТЗЫВ

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

Below the schedule, there's a blog post titled 'Углубленный Python' (Advanced Python) with the subtitle 'Blog for materials on the "Advanced Python" course'. It has 57 readers and 2 topics. A search bar and a 'Найти' (Find) button are below the post. To the right, there's a sidebar for 'Прямой эфир' (Live Stream) and a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

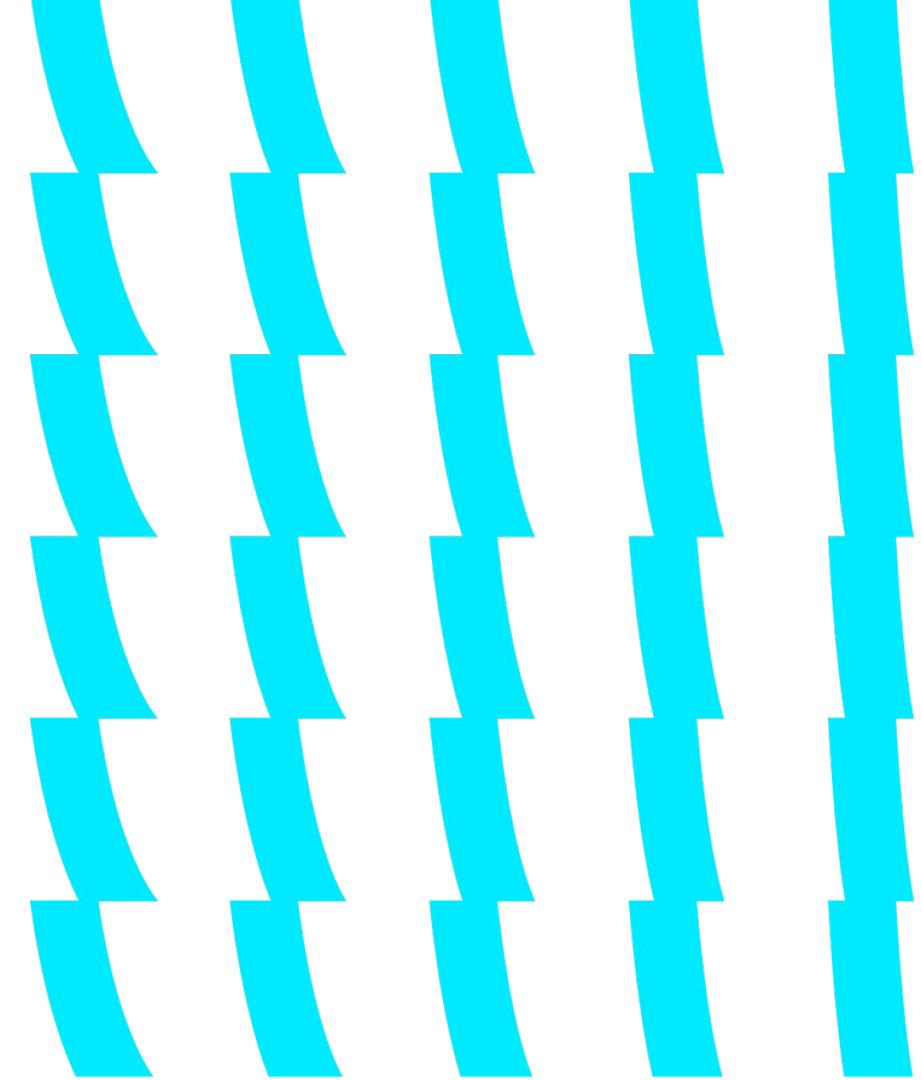
Квиз про прошлой лекции



## Содержание занятия

1. Потоки, GIL
2. Процессы
3. IPC
4. subprocess

# **Потоки (Threads)**



## Потоки

**Thread (поток)** - это сущность операционной системы, процесс выполнения на процессоре набора инструкций, а именно программного кода.

# Потоки: создание и запуск

```
class threading.Thread(  
    group=None, target=None, name=None, args=(), kwargs={}, *, daemon=None  
)
```

- th.start()
- th.join(timeout=None)
- th.run()
- th.is\_alive()
- th.name
- th.ident
- th.native\_id
- th.daemon

# Потоки: создание и запуск

```
import threading

1. class CustomThread(threading.Thread):
    def run(self):
        func()

    th = CustomThread()

2. th = threading.Thread(target=func)

th.start()
th.join()
```

## Потоки



# ПОТОКИ: СИНХРОНИЗАЦИЯ

- `threading.Lock`
- `threading.RLock`
- `threading.Semaphore`
- `threading.BoundedSemaphore`
- `threading.Event`
- `threading.Timer`
- `threading.Barrier`

Дополнительно:

`queue (Queue, LifoQueue, PriorityQueue)`

## Потоки: local

```
import threading  
  
my_data = threading.local()  
my_data.x = 42
```

## GIL

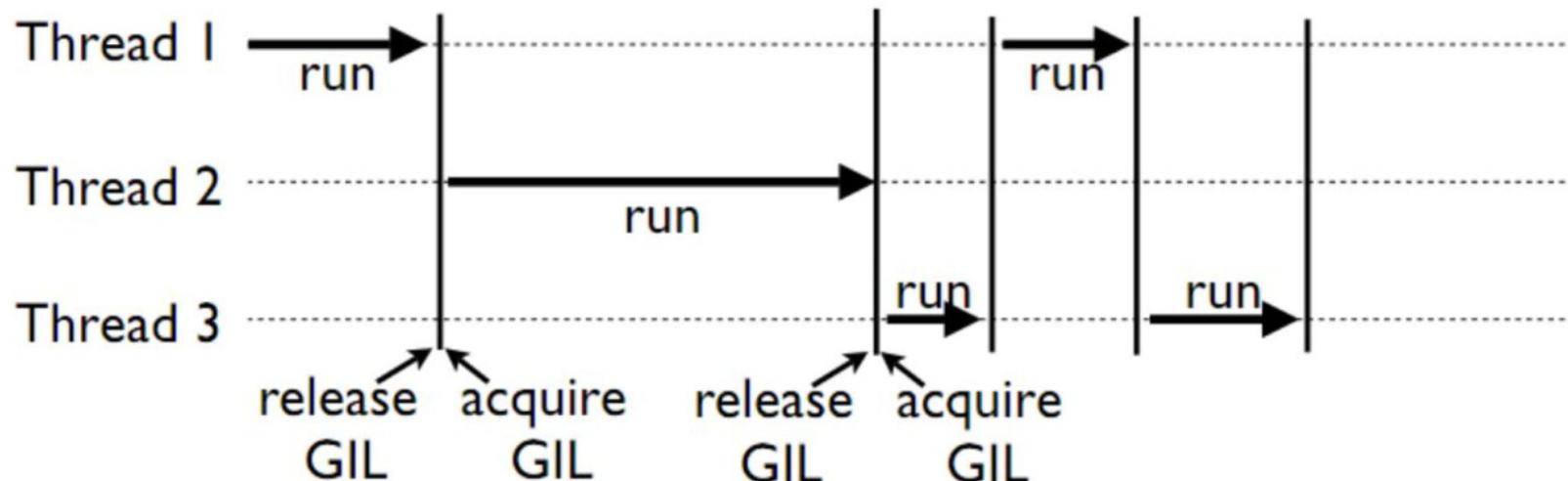
**Global Interpreter Lock (GIL)** — это способ синхронизации потоков, который используется в некоторых интерпретируемых языках программирования.

Mutex, который разрешает только одному потоку использовать интерпретатор python

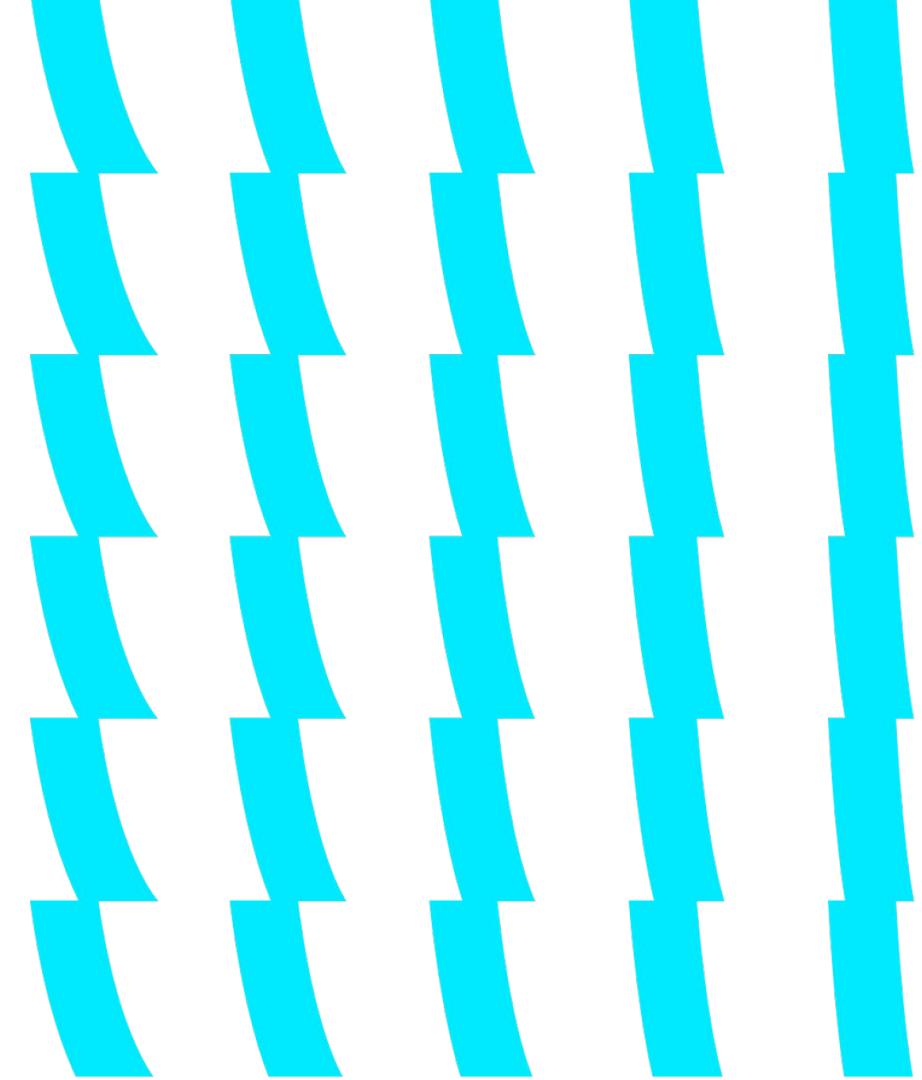
# GIL

- Что решает? - Race conditions
- Почему глобальный? - Deadlocks, производительность
- Выбран в качестве решения из-за C extention
- Изначально вводился для I/O bound потоков

# GIL



# Multiprocessing



# Multiprocessing

**Процесс** - абстракция, которая инкапсулирует в себе все ресурсы процесса: открытые файлы, отображенные в память файлы, дескрипторы, потоки и тд.

Составные части:

1. Образ машинного кода;
2. Область памяти, в которую включается исполняемый код, данные процесса (входные и выходные данные, стек вызовов и куча для хранения динамически создаваемых данных);
3. Дескрипторы ОС, например, файловые;
4. Состояние процесса.

# Multiprocessing

```
import os
from multiprocessing import Process

def print_info(name):
    print(f"Process {name}, pid={os.getpid()}, parent pid={os.getppid()}")

if __name__ == "__main__":
    print_info("main")
    processes = [
        Process(target=print_info, args=(f"child{i}",))
        for i in range(1, 5)
    ]
    for proc in processes:
        proc.start()
    for proc in processes:
        proc.join()
```

# Multiprocessing: Pool

```
import multiprocessing
import time

def countdown(n):
    while n > 0:
        n -= 1

if __name__ == '__main__':
    t1 = time.time()
    with multiprocessing.Pool(2) as p:
        p.apply_async(countdown, (100000000,))
        p.apply_async(countdown, (100000000,))
    p.close()
    p.join()
    t2 = time.time()
    print(t2 - t1)
```

# Multiprocessing: синхронизация

- Lock, Semaphore, Event и тп
- Value

```
result = multiprocessing.Value("i")
```

- Array

```
result = multiprocessing.Array("i", 4)
```

- Manager

```
with multiprocessing.Manager() as manager:  
    records = manager.list([])
```

- Queue

```
q = multiprocessing.Queue()
```

- Pipe

```
parent_conn, child_conn = multiprocessing.Pipe()
```

# **IPC**

**Inter Process Communications**  
**(межпроцессное взаимодействие)**

# IPC

ОС предоставляют механизмы для IPC:

- механизмы обмена сообщениями
- механизмы синхронизации
- механизмы разделения памяти
- механизмы удаленных вызовов (RPC)

## IPC: виды

- файл
- сигнал
- сокет
- каналы (именованные/неименованные)
- семафор
- разделяемая память
- обмен сообщениями
- проецируемый в памяти файл
- очередь сообщений
- почтовый ящик

# IPC: СИГНАЛЫ

```
import os, time, signal

def signal_handler(signal_num, frame):
    print(f"Handle signal {signal_num}")

if __name__ == "__main__":
    signal.signal(signal.SIGUSR1, signal_handler)
    signal.signal(signal.SIGUSR2, signal_handler)

    print(f"pid={os.getpid()}")
    while True:
        time.sleep(0.5)
```

## IPC: сокеты

```
import socket

server = socket.socket(socket.AF_UNIX, socket.SOCK_DGRAM)
server.bind("/tmp/py_unix_example")
data = server.recv(1024)

client = socket.socket(socket.AF_UNIX, socket.SOCK_DGRAM)
client.connect("/tmp/py_unix_example")
client.send(data.encode())
```

# IPC: каналы (pipe)

```
# sender.py

import os

fpath = "/tmp/example fifo"
os.mkfifo(fpath)

fifo = open(fpath, "w")
fifo.write("Hello!\n")
fifo.close()
```

```
# receiver.py

import os
import sys

fpath = "/tmp/example fifo"

fifo = open(fpath, "r")
for line in fifo:
    print(f"Recv: {line}")
fifo.close()
```

## IPC: mmap

```
import mmap

with open("data.txt", "w") as f:
    f.write("Hello, python!\n")

with open("data.txt", "r+") as f:
    map = mmap.mmap(f.fileno(), 0)
    print(map.readline()) # Hello, python!
    print(map[:5]) # Hello
    map[7:] = "world!\n"
    map.seek(0)
    print(map.readline()) # Hello, world!
    map.close()
```

# subprocess

```
subprocess.run(args, **kwargs)
```

```
subprocess.Popen(args, **kwargs)
```

## Домашнее задание #6

- Клиент и сервер для обкачки урлов и подсчета наиболее употребимых слов
- +тесты
- flake8 + pylint перед сдачей

# Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

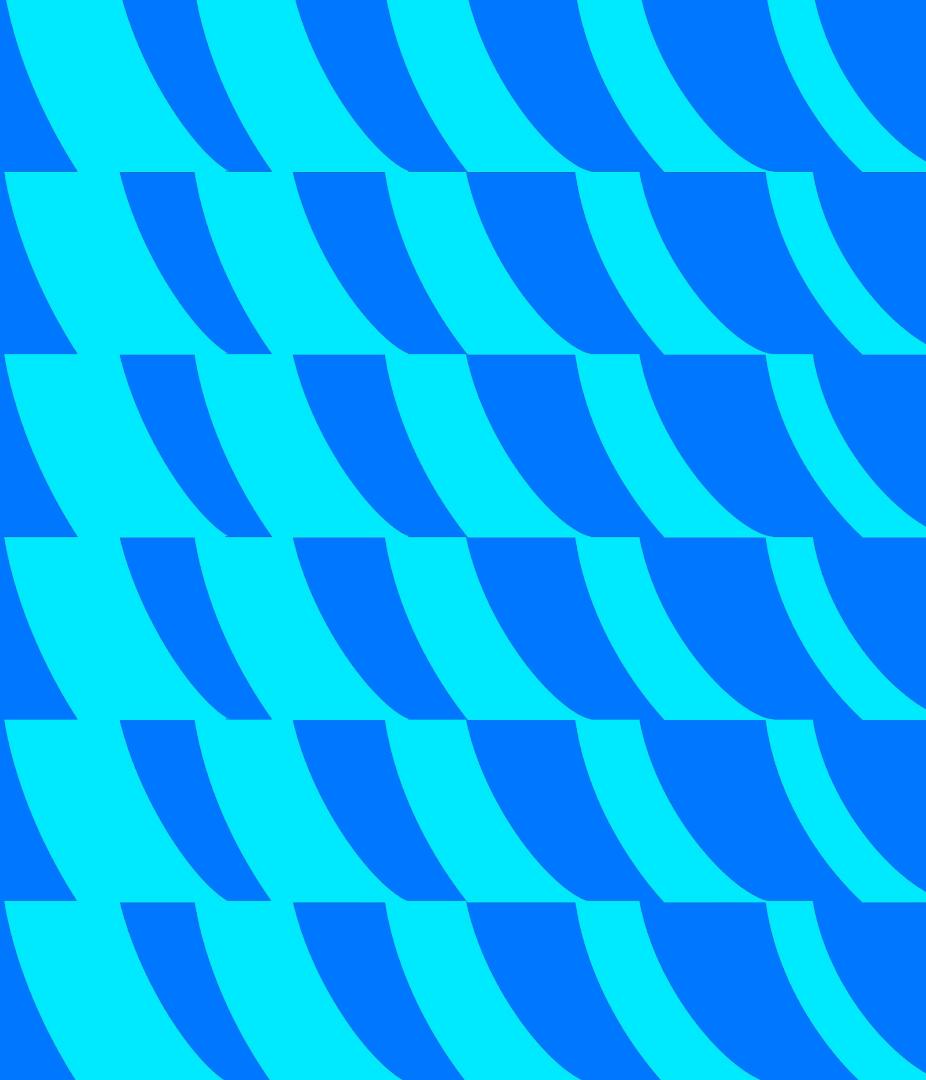
Below the schedule, a blog post titled 'Углубленный Python' is shown. It has 57 readers and 2 topics. There are buttons for 'Подписаться' (Subscribe) and 'Создать топик' (Create topic). A search bar and a 'Найти' (Find) button are also present.

On the right side, there's a sidebar for 'Прямой эфир' (Live Stream) with a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

Спасибо за  
внимание



образование



# C-extensions

Антон Кухтичев

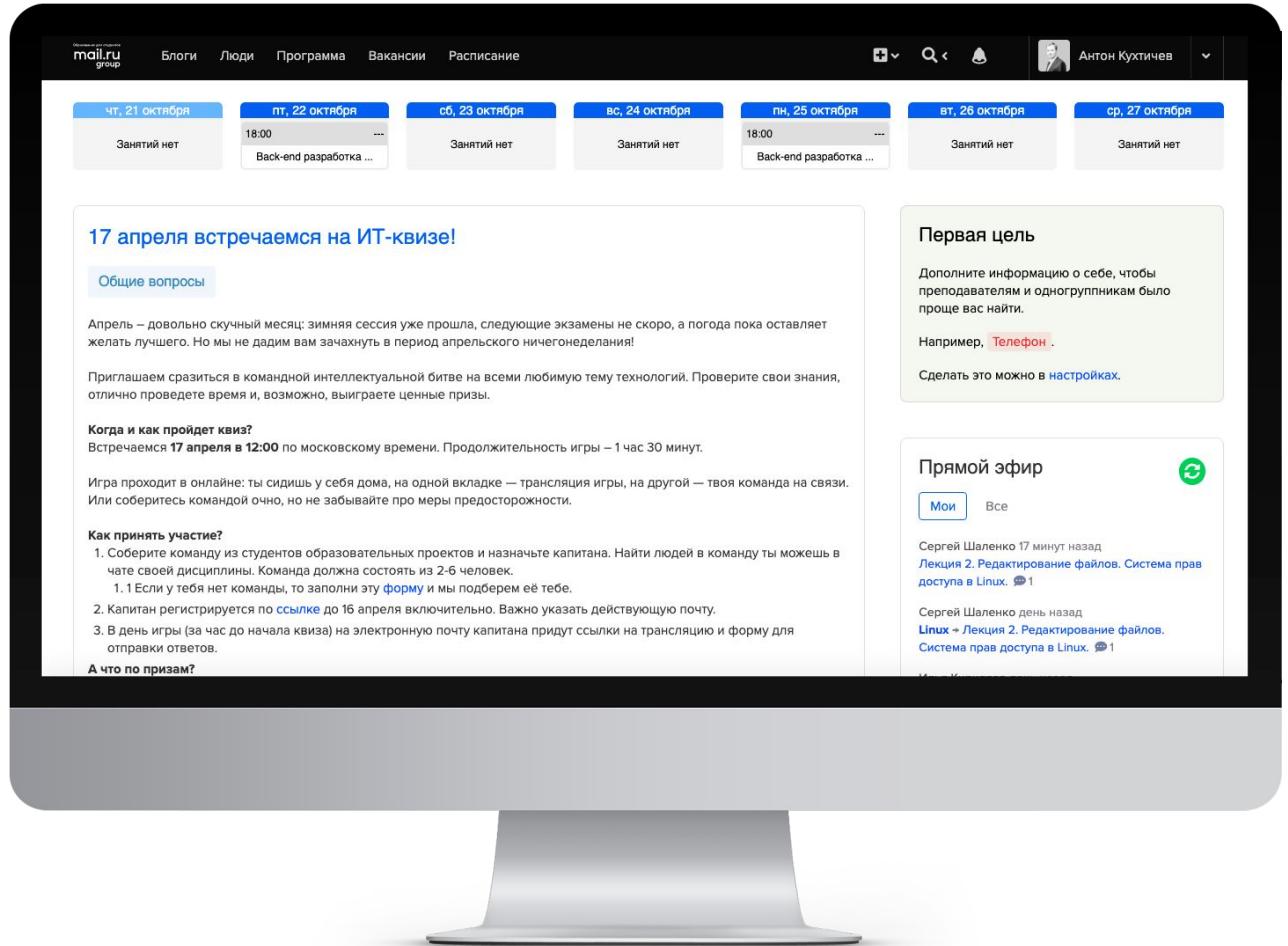


## Содержание занятия

- `ctypes`;
- `cffi`;
- C API;
- Cython

# Напоминание отметиться на портале

## и оставить отзыв после лекции



# Для чего это всё???

- Вам нужна скорость и вы знаете, что С в X раз быстрее Python;
- Вам нужна конкретная С-библиотека и вы не хотите писать “велосипед” на Python;
- Вам нужен низкоуровневый интерфейс управления ресурсами для работы с памятью и файлами;
- Просто потому что Вам так хочется

# ctypes

# ctypes

- Работает с DLL (Dynamic link library);
- ctypes определяет типы данных, совместимых с языком С:
  - c\_bool
  - c\_char
  - c\_int
  - c\_char\_p
  - c\_void\_p
- Чтобы подключить библиотеку нужно либо вызвать
  - `ctypes.cdll.LoadLibrary('<dll path>')`
  - `ctypes.CDLL('<dll path>')`

# ctypes

```
int sum(int *arr, int len)
{
    int res = 0;
    for (int i = 0; i < len; ++i)
    {
        res += arr[i];
    }
    return res;
}
```

```
gcc -fPIC -shared -o sumlib.so 1.c
```

# ctypes

```
import ctypes
from typing import List

lib1 = ctypes.CDLL('./lib1.so')
lib1.sum.argtypes = (ctypes.POINTER(ctypes.c_int), ctypes.c_int)
```

# ctypes

```
def sum(arr: List[int]) -> int:  
    arr_len = len(arr)  
    arr_type = ctypes.c_int * arr_len  
    result = lib1.sum(arr_type(*arr), ctypes.c_int(arr_len))  
    return int(result)
```

# CFFI

C Foreign Function Interface

# CFFI

```
# Установка  
pip install cffi
```

CFFI (C Foreign Function Interface) генерирует поверх нашей библиотеки свою обвязку и компилирует её в библиотеку с которой мы и будем работать.

# CFFI

```
from cffi import FFI

ffi = FFI()
lib = ffi.dlopen('../ctypes/lib1.so')

ffi.cdef('''int sum(int* arr, int len);'''')
arr = [1, 2, 3, 4]
c_arr = ffi.new('int[]', arr)

s = lib.sum(c_arr, len(arr))
print(s)
```

# CFI

```
#include <stdlib.h>

struct Point {
    int x;
    int y;
};

int area(struct Point *p1, struct Point *p2) {
    return abs((p2->y - p1->y) * (p1->x - p2->x));
}

gcc -fPIC -shared -o lib2.so 2.c
```

# CFI

```
from cffi import FFI

ffi = FFI()
lib = ffi.dlopen('./lib2.so')

ffi.cdef('''
struct Point {
    int x;
    int y;
};
int area(struct Point *p1, struct Point *p2);
'''')
```

# FFI

```
p1 = ffi.new('struct Point*')
```

```
p2 = ffi.new('struct Point*')
```

```
p1.x = 0
```

```
p1.y = 0
```

```
p2.x = 10
```

```
p2.y = 10
```

```
s = lib.area(p1, p2)
```

```
print(s)
```

# Плюсы и минусы CFFI

Плюсы:

- простой синтаксис при использовании в Python;
- не нужно перекомпилировать исходную библиотеку.

Минусы:

- не удобная сборка, нужно прописывать пути до всех заголовочных файлов и библиотек;
- создается еще одна динамическая библиотека, которая использует исходную.

# C Extensions

# c extensions

```
>>> import spam  
>>> status = spam.system("ls -l")
```

# c extensions

```
static PyObject* spam_system(PyObject *self, PyObject *args)
{
    const char *command;
    int sts;

    if (!PyArg_ParseTuple(args, "s", &command))
        return NULL;
    sts = system(command);
    if (sts < 0) {
        PyErr_SetString(SpamError, "System command failed");
        return NULL;
    }
    return PyLong_FromLong(sts);
}
```

# Cython

# Cython

```
# Установка  
pip install cython
```

При работе с функциями нам доступны следующие типы:

- `def` – обычная Python-функция, вызывается только из Python.
- `cdef` – Cython-функция, которую нельзя вызвать из обычного Python-кода. Такие функции можно вызывать только в пределах Cython-кода.
- `cpdef` – Функция, доступ к которой можно получить и из C, и из Python.

# Cython

```
# setup.py
from setuptools import setup, Extension
from Cython.Build import cythonize

setup(
    ext_modules= cythonize(['cutils.pyx'])
)

# Выполним компиляцию
$ python setup.py build_ext --inplace
```

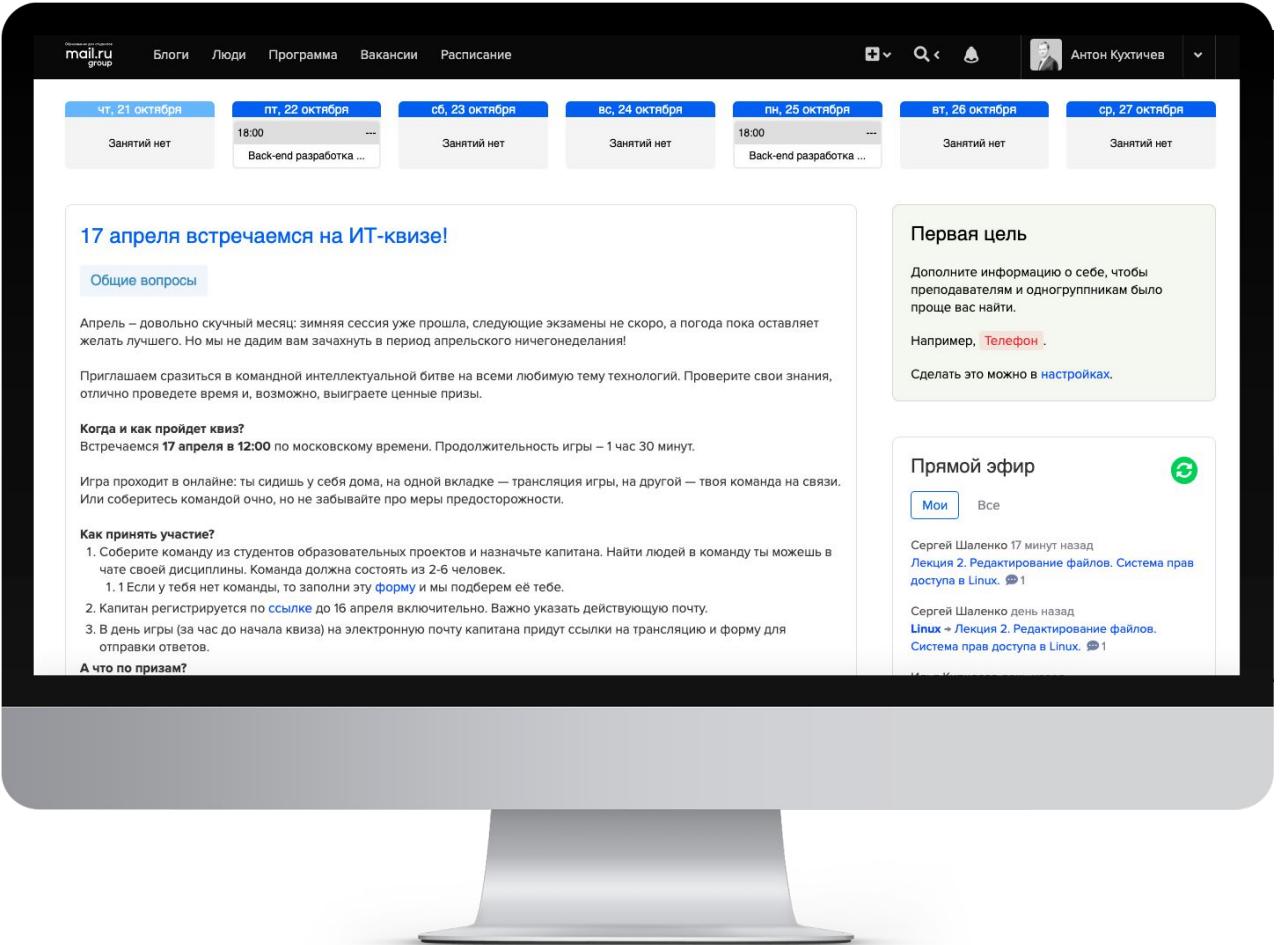
# Домашнее задание

- Реализовать умножение цепочки матриц на C и сравнить производительность кода на C и на Python.

Литература: [Expert Python Programming - Second edition](#)

# Напоминание оставить отзыв

Это правда важно



Спасибо за  
внимание!

Вопросы?

# Углубленный Python

Лекция 8

## Асинхронное программирование

Кандауров Геннадий



образование

# Напоминание отметиться на портале

+ ОСТАВИТЬ ОТЗЫВ

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

Below the schedule, there's a blog post titled 'Углубленный Python' (Advanced Python) with the subtitle 'Blog for materials on the "Advanced Python" course'. It has 57 readers and 2 topics. A search bar and a 'Найти' (Find) button are below the post. To the right, there's a sidebar for 'Прямой эфир' (Live Stream) and a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

Квиз про прошлой лекции



# Содержание занятия

1. Цикл событий
2. Корутины, нативные корутины
3. `asyncio`

# subprocess

```
subprocess.Popen(args, **kwargs)
```

```
subprocess.run(args, **kwargs)
```

```
subprocess.run(["ls", "-l", "/dev/null"], capture_output=True)
```

```
CompletedProcess(args=['ls', '-l', '/dev/null'], returncode=0,
```

```
stdout=b'crw-rw-rw- 1 root root 1, 3 Jan 23 16:23 /dev/null\n', stderr=b'')
```

# **Асинхронное программирование**

## Блокирующие операции

- connect, accept, recv, send - блокирующие операции
- C10k problem, <http://kegel.com/c10k.html>
- Потоки дорого стоят (CPU & RAM)
- Потоки простояивают часть времени

# Блокирующие операции

```
import socket
server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_sock.bind(("localhost", 15000))
server_sock.listen()

while True:
    client_sock, addr = server_sock.accept()
    while True:
        data = client_sock.recv(4096)
        if not data:
            break
        else:
            client_sock.send(data.decode().upper().encode())
    client_sock.close()
```

# Неблокирующие операции

Системные вызовы:

- select (`man 2 select`)
- poll (`man 2 poll`)
- epoll (`man 7 epoll`)
- kqueue

python:

- `select`
- `selectors`

# select

```
from select import select

def event_loop():
    while True:
        ready_to_read, _, _ = select(to_monitor, [], [])

        for sock in ready_to_read:
            if sock is server_sock:
                accept_conn(sock)
            else:
                respond(sock)
```

# selectors

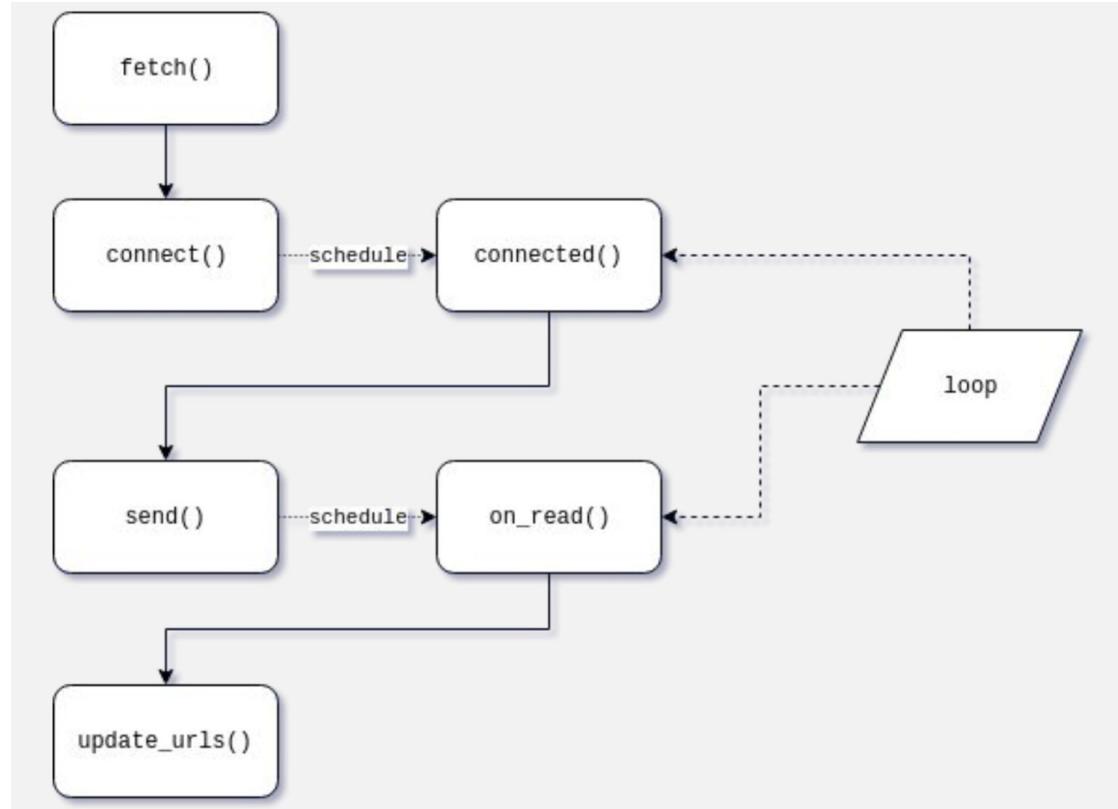
```
import selectors

selector = selectors.DefaultSelector()
selector.register(server_sock, selectors.EVENT_READ, accept_conn)

def event_loop():
    while True:
        events = selector.select() # (key, events_mask)

        for key, _ in events:
            # key: NamedTuple(fileobj, events, data)
            callback = key.data
            callback(key.fileobj)
            # selector.unregister(key.fileobj)
```

# Callback hell



# Generator based event loop

Дэвид Бизли (David Beazley), "Python Concurrency From the Ground Up: LIVE!"

```
def event_loop():
    while any([tasks, to_read, to_write]):
        while not tasks:
            ready_to_read, ready_to_write, _ = select(to_read, to_write, [])
            for sock in ready_to_read:
                tasks.append(to_read.pop(sock))
            for sock in ready_to_write:
                tasks.append(to_write.pop(sock))
        try:
            task = tasks.pop(0)
            op_type, sock = next(task)
            if op_type == "read":
                to_read[sock] = task
            elif op_type == "write":
                to_write[sock] = task
        except StopIteration:
            pass
```

# Корутины

```
def grep(pattern):
    print("start grep for", pattern)
    while True:
        s = yield
        if pattern in s:
            print("found!", s)
        else:
            print("no %s in %s" % (pattern, s))

g = grep("python")
next(g)
g.send("data")
g.send("deep python")

$ python grep_python.py
start grep for python
no python in data
found! deep python
```

# Корутины

- использование `yield` более обобщенно определяет корутину
- не только генерируют значения
- потребляют данные, отправленные через `.send`
- отправленные данные возвращаются через `data = yield`

# Нативные корутины

## **coroutine**

Coroutines are a more generalized form of subroutines. Subroutines are entered at one point and exited at another point. Coroutines can be entered, exited, and resumed at many different points.

They can be implemented with the `async def` statement.

See also [PEP 492](#).

# Нативные корутины

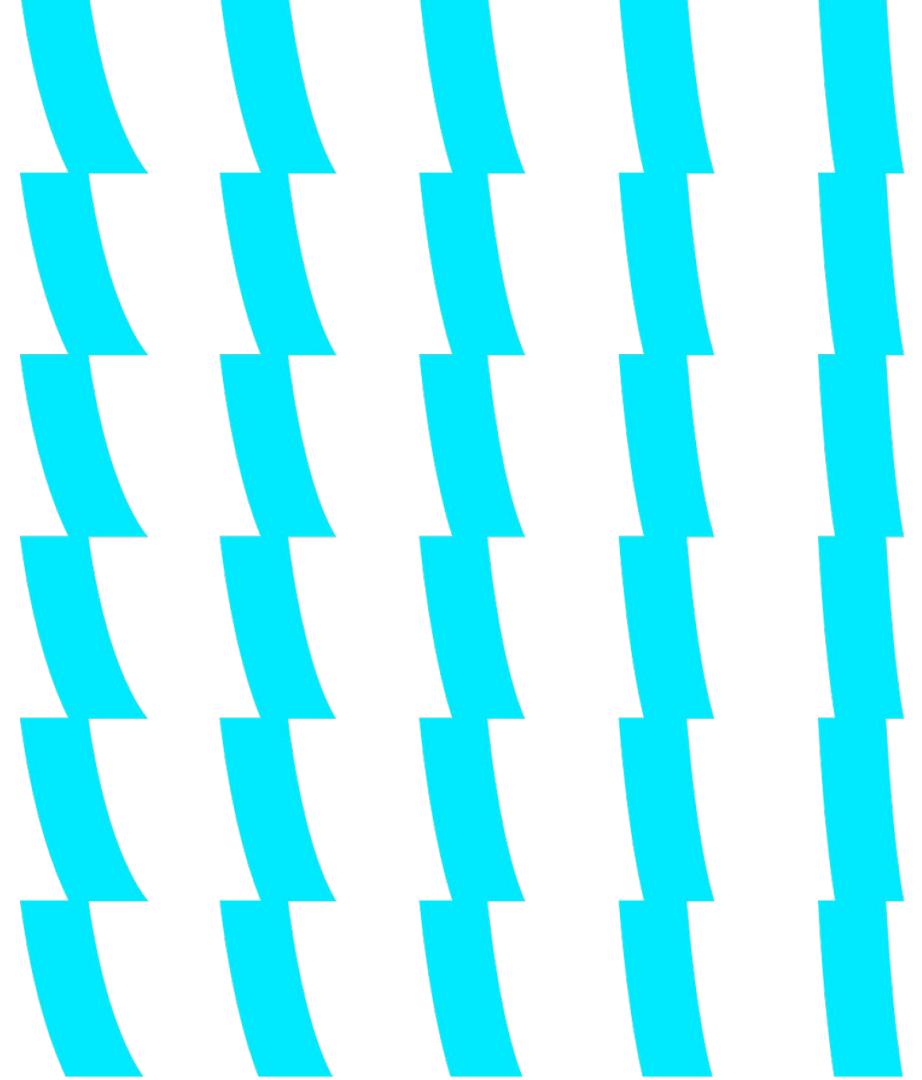
```
async def say_after(delay, what):
    await asyncio.sleep(delay)
    print(what)

async def main():
    print(f"started at {time.strftime('%X')}")
    await say_after(1, 'hello')
    await say_after(2, 'world')
    print(f"finished at {time.strftime('%X')}")

asyncio.run(main())
```

```
>run.py
started at 16:42:46
hello
world
finished at 16:42:49
```

# asyncio



# asyncio

- 1 процесс
- 1 поток
- кооперативная многозадачность (vs вытесняющая)
- передача управления в event loop на ожидающих операциях
- `async/await` это API Python, а не часть asyncio

# asyncio

Event loop:

coroutine > Task (Future)

- **Future** представляет ожидаемый в будущем (eventual) результат асинхронной операции;
- **Task** это **Future-like** объект, запускающий корутины в событийном цикле;
- **Task** используется для запуска нескольких корутин в событийном цикле параллельно.

# asyncio

## High-level APIs

- Coroutines and Tasks
- Streams
- Synchronization Primitives
- Subprocesses
- Queues
- Exceptions

# asyncio

## Low-level APIs

- Event Loop
- Futures
- Transports and Protocols
- Policies
- Platform Support

# asyncio

## Вспомогательное API

- `asyncio.create_task`
- `asyncio.sleep`
- `asyncio.gather`
- `asyncio.shield`
- `asyncio.wait_for`
- `asyncio.wait`
- `asyncio.Queue`
- `asyncio.Lock`
- `asyncio.Event`

## **Домашнее задание #8**

- Асинхронный сервер для равномерной обкачки веб-страниц
- +тесты
- flake8 + pylint перед сдачей

# Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

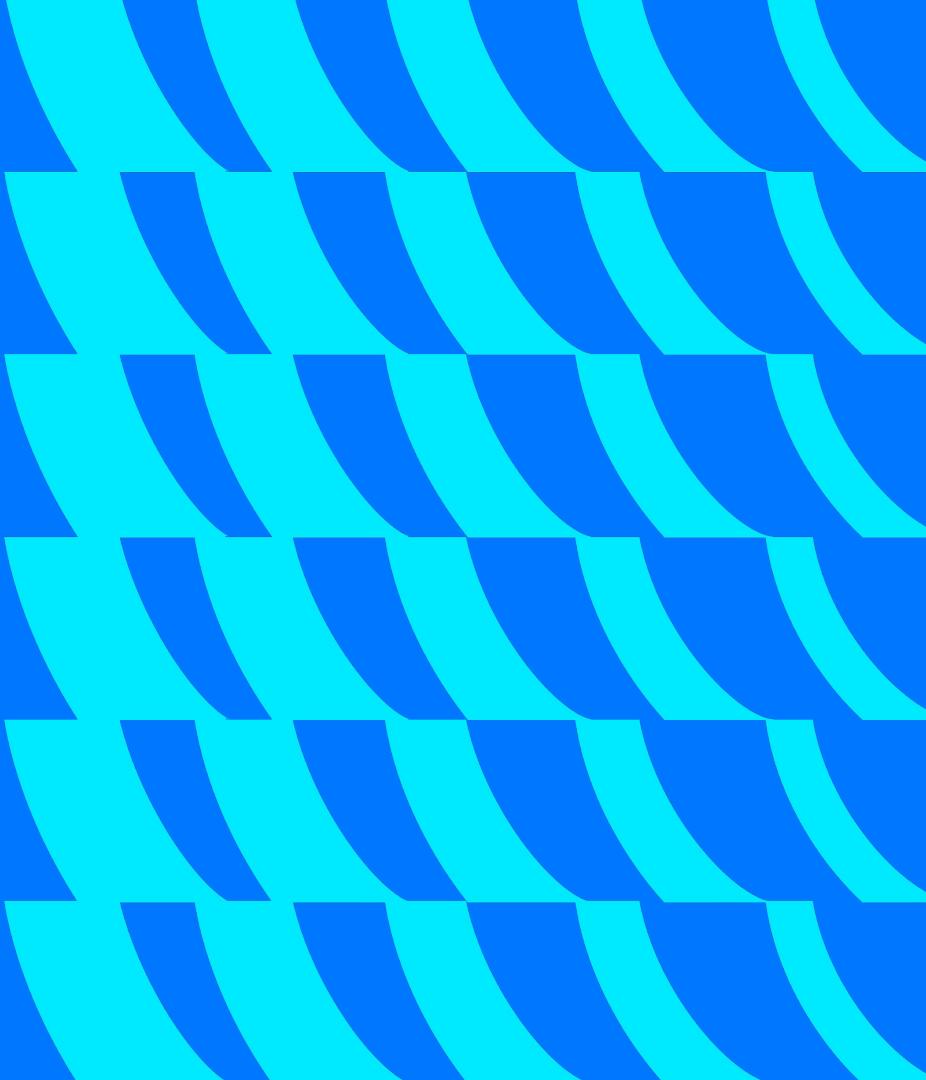
Below the schedule, a blog post titled 'Углубленный Python' is shown. It has 57 readers and 2 topics. There are buttons for 'Подписаться' (Subscribe) and 'Создать топик' (Create topic). A search bar and a 'Найти' (Find) button are also present.

On the right side, there's a sidebar for 'Прямой эфир' (Live Stream) with a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

Спасибо за  
внимание



образование



# Углубленный Python

## Лекция 9

### Управление памятью, профилирование

Кандауров Геннадий



образование

# Напоминание отметиться на портале

+ ОСТАВИТЬ ОТЗЫВ

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

In the center, there's a blog post titled 'Углубленный Python' (Advanced Python) with a description: 'Blog for materials on the "Advanced Python" course'. It has 57 readers and 2 topics. Below the post is a search bar and a 'Найти' (Find) button. To the right, there's a sidebar for 'Прямой эфир' (Live Stream) and a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

Квиз про прошлой лекции



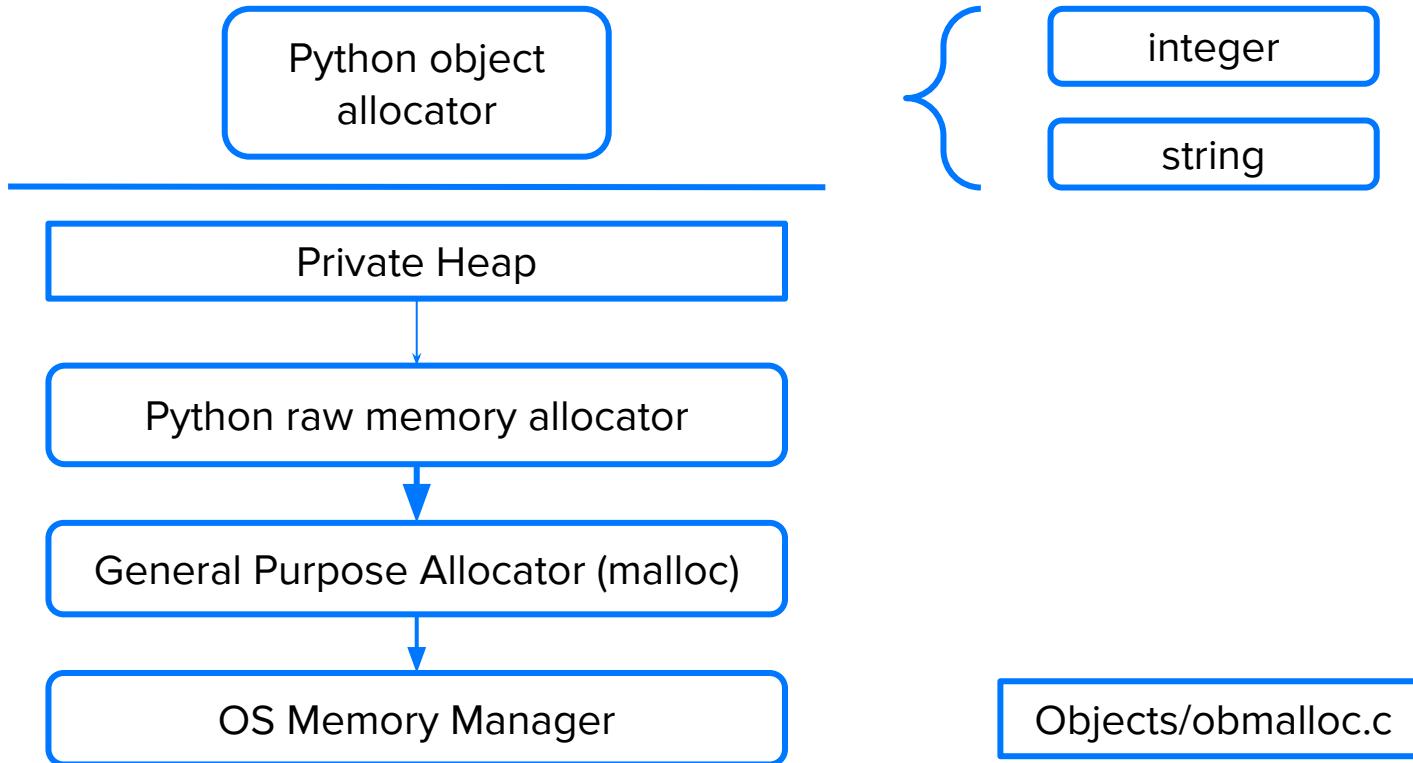
## Содержание занятия

1. Устройство памяти
2. Счетчик ссылок, gc
3. Оптимизации (slots, weakref)
4. Профилирование

# **Устройство памяти**

Выделение, освобождение, управление

# Python memory manager

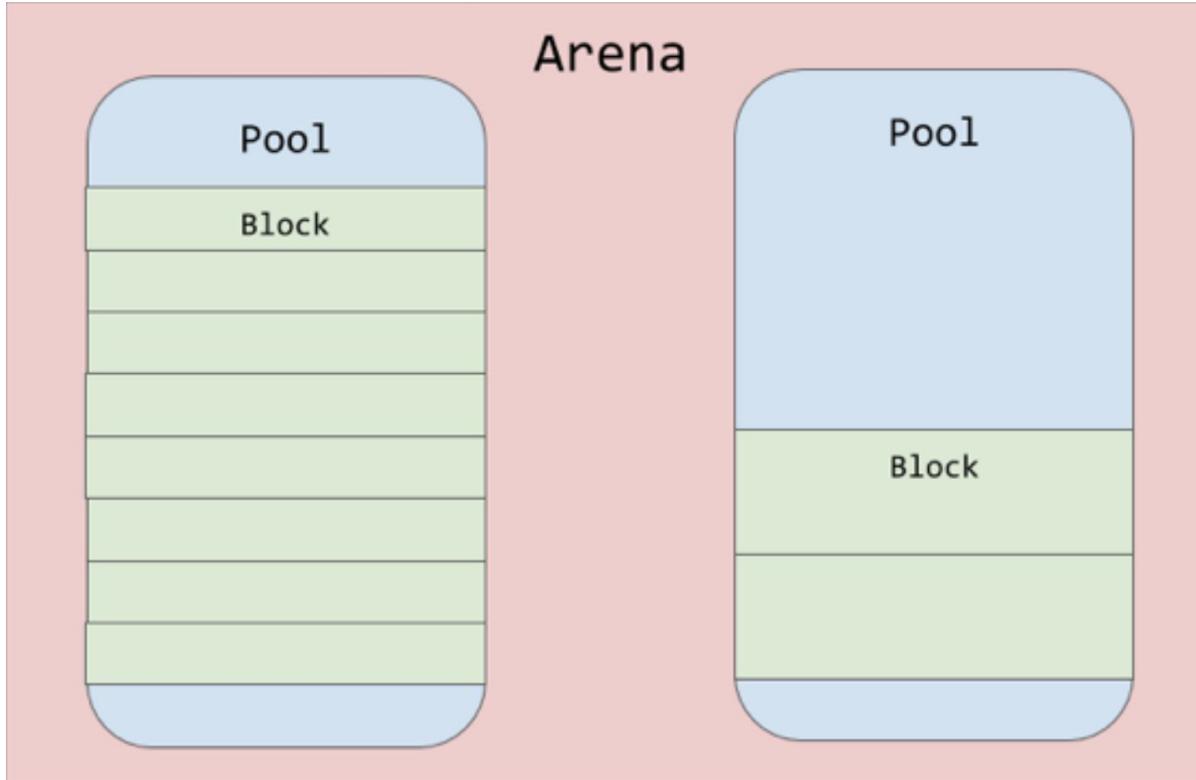


# Выделение памяти

- Большие объекты (> 512 байт): С allocator;
- Меньшие объекты (<= 512 байт): арены, пулы, блоки;
  - Блок хранит один объект от 1 до 512 байт;
  - Пул хранит блоки, занимает одну страницу памяти (4Кб);
  - Аренда хранит пулы, занимает 256Кб;

Только аренда может освобождать память

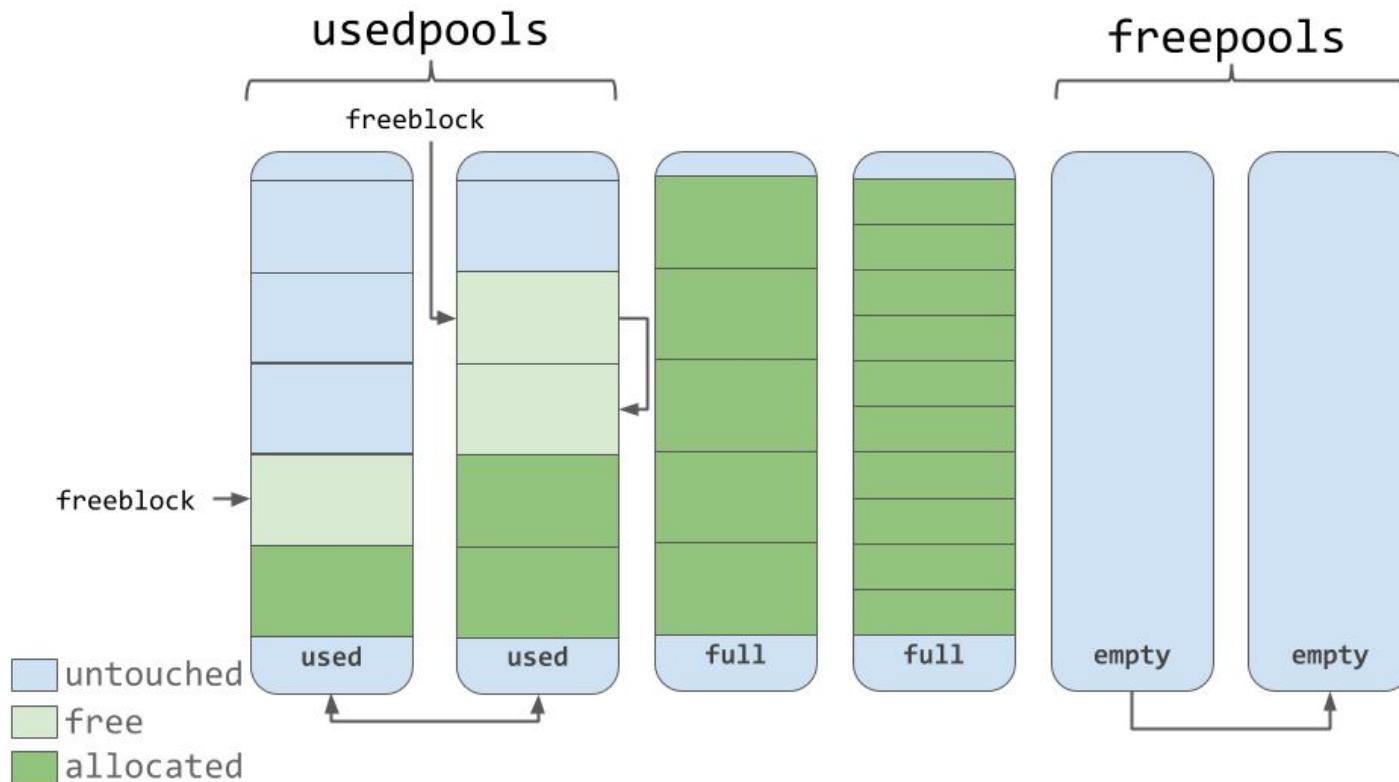
# Выделение памяти



# Выделение памяти

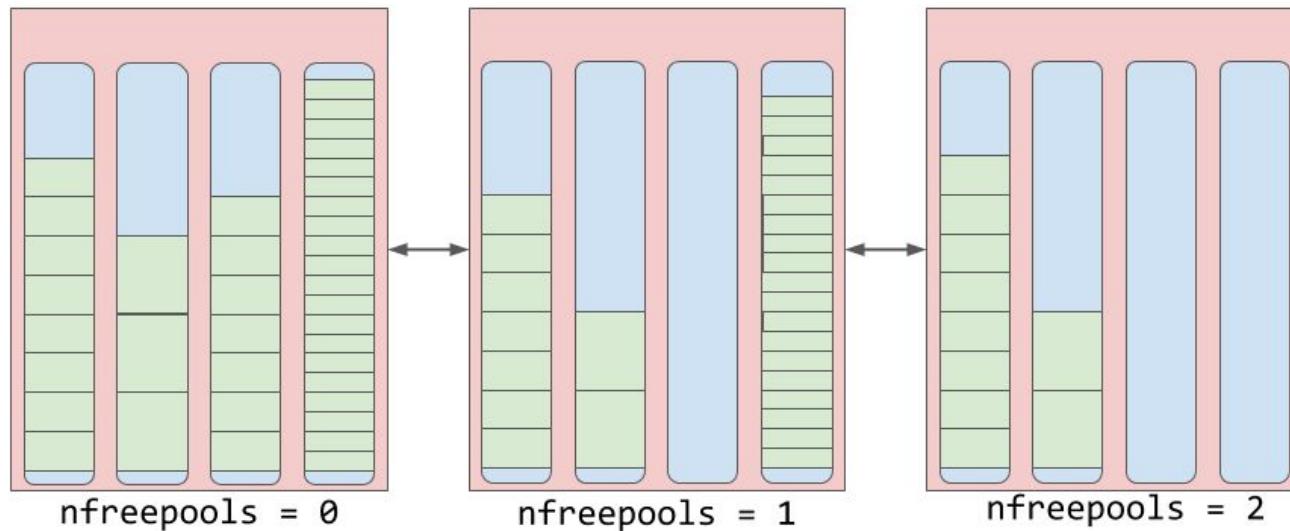
* Request in bytes	Size of allocated block	Size class idx
*	-----	
* 1–8	8	0
* 9–16	16	1
* 17–24	24	2
* 25–32	32	3
* 33–40	40	4
* 41–48	48	5
* 49–56	56	6
* 57–64	64	7
* 65–72	72	8
* ...	...	...
* 497–504	504	62
* 505–512	512	63
*		
* 0, SMALL_REQUEST_THRESHOLD + 1 and up:	routed to the underlying	
* allocator.		

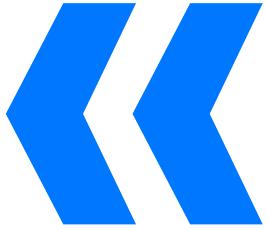
# Выделение памяти



# Выделение памяти

usable\_arenas





*The only reliable way to free memory is to  
terminate the process.*

**Счетчик ссылок, гс**

# PyObject

```
typedef struct _object {  
    _PyObject_HEAD_EXTRA  
    Py_ssize_t ob_refcnt;  
    PyTypeObject *ob_type;  
} PyObject;
```

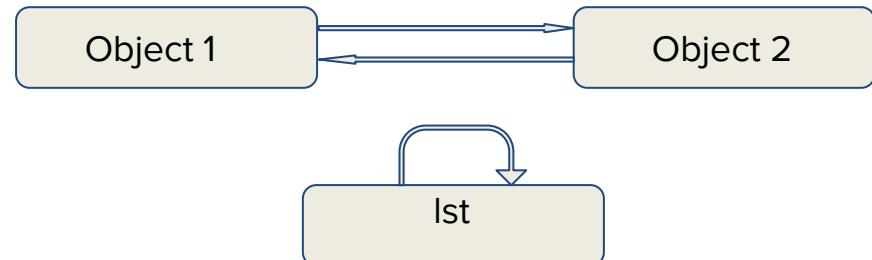
# Освобождение памяти

- счетчик ссылок, refcounter
- generational garbage collector, модуль gc (можно отключить)

```
>>> import sys  
>>> foo = []  
>>> print(sys.getrefcount(foo))  
>>> def bar(a):  
        print(sys.getrefcount(a))  
>>> bar(foo)  
>>> print(sys.getrefcount(foo))
```

# Счетчик ссылок

- + Память сразу можно очистить
- Циклические ссылки
- Блокирование потоков
- Доп расход CPU и RAM

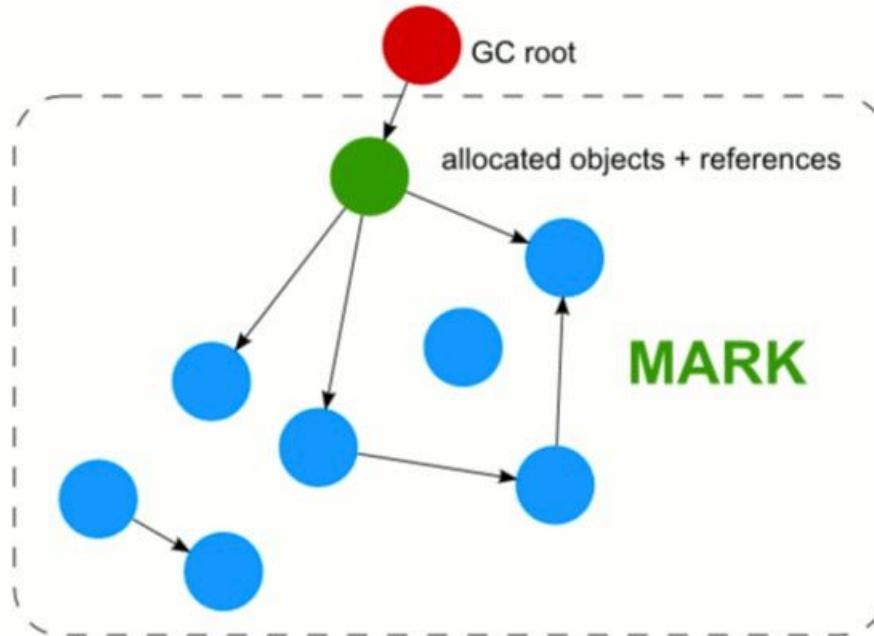


## Сборщик мусора

- + Не нужно думать об очистке памяти
- + Никаких double free ошибок
- + Решение проблем утечек памяти
- Доп расход CPU и RAM
- Момент сборки мусора непредсказуем

# Mark and sweep gc

## Mark and sweep (MARK)



# Сборщик мусора

**gc** (<https://docs.python.org/3/library/gc.html>)

следит только за объектами контейнерами, если они содержат тоже объекты-контейнеры:

- `list`
- `dict`
- `tuple`
- `class`
- `etc`

## Сборщик мусора: управление

- Включение/выключение gc  
`gc.enable()`, `gc.disable()`, `gc.isenabled()`
- Запуск сборки мусора  
`gc.collect(generation=2)`
- Получение всех объектов  
`gc.get_objects(generation=None)`

# weakref

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

- `weakref.ref`
- `WeakKeyDictionary`, `WeakValueDictionary`, `WeakSet`,  
`WeakMethod`
- `finalize`
- `getweakrefcount(obj)`, `getweakrefs(obj)`
- `list`, `dict`: только для подклассов
- `tuple`, `int`: не поддерживаются

# weakref

```
>>> import weakref
>>> class Object:
...     pass
...
>>> obj1 = Object()
>>> ref = weakref.ref(obj1)
>>> obj2 = ref()
>>> obj1 is obj2
True
>>> del obj1, obj2
>>> print(ref())
None
```

# slots

## `object.__slots__`

Позволяет явно указать поля, которые будут в классе.

В случае указания `__slots__` пропадают поля `__dict__` и `__weakref__`.

Используя `__slots__` можно экономить на памяти и времени доступа к атрибутам объекта.

```
class Point:  
    __slots__ = ("x", "y")  
  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y
```

# Профилирование

Сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (обычно подпрограмм), число верно предсказанных условных переходов, число кэш-промахов и т. д.

# Профилирование

Цель:

- найти узкие места в коде

Основные способы:

- CPU
- Память
- Частота/продолжительность вызовов функций

Методы:

- Статистический (сэмплирование)
- Детерминированный (инструментирование)

# Профилирование

- `cProfile` - написанная на C, быстрая реализация профилировщика
- `profile` - нативная реализация профилировщика на чистом python, значительно медленнее

```
python -m cProfile -o output.txt ptest.py
```

```
import pstats
p = pstats.Stats("output.txt")
p.strip_dirs().sort_stats(-1).print_stats()
```

# Профилирование

```
import cProfile, pstats, io

pr = cProfile.Profile()
pr.enable()
# ... do something ...
pr.disable()

s = io.StringIO()
sortby = "cumulative"
ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
ps.print_stats()
print(s.getvalue())
```

# cProfile

```
1567629 function calls (1166637 primitive calls) in 809.730 seconds

Ordered by: cumulative time

      ncalls  tottime  percall  cumtime  percall filename:lineno(function)
           1    0.164    0.164  809.738  809.738 /Users/project/src/.env3/lib/python3.7/site-packages/tornado/ioloop.py:568(start)
  4961   806.444    0.163  806.444    0.163 /Users/project/src/.env3/lib/python3.7/site-packages/tornado/platform/kqueue.py:66(poll)
9982/8005    0.086    0.000    3.095    0.000 /Users/project/src/.env3/lib/python3.7/site-packages/tornado/stack_context.py:269(wrapped)
  5657    0.011    0.000    2.767    0.000 /Users/project/src/.env3/lib/python3.7/site-packages/tornado/ioloop.py:471(_run_callback)
6766/2479    0.083    0.000    1.869    0.001 /Users/project/src/.env3/lib/python3.7/site-packages/tornado/gen.py:507(run)
  2445    0.009    0.000    1.775    0.001 /Users/project/src/.env3/lib/python3.7/site-packages/tornado/gen.py:567(inner)
  2445    0.005    0.000    1.764    0.001 /Users/project/src/.env3/lib/python3.7/site-packages/tornado/gen.py:497(set_result)
   430    0.008    0.000    0.902    0.002 /Users/project/src/gekko/net/resolver.py:414(resolve)
    75    0.000    0.000    0.669    0.009 /Users/project/src/gekko/handlers2/executor.py:93(callback)
    75    0.000    0.000    0.669    0.009 /Users/project/src/gekko/handlers2/executor.py:72(_handler_callback)
     48    0.000    0.000    0.669    0.014 /Users/project/src/gekko/handlers2/executor.py:114(_done)
    72    0.000    0.000    0.612    0.009 /Users/project/src/gekko/location2.py:266(_call_location_method)
    60    0.000    0.000    0.610    0.010 /Users/project/src/gekko/location2.py:91(create_gen_tasks)
    63    0.000    0.000    0.609    0.010 /Users/project/src/gekkoapps/gosearch/locations/ajax_web.py:27(get)
     9    0.000    0.000    0.576    0.064 /Users/project/src/gekkoapps/common/locations/base.py:104(create_response)
     9    0.001    0.000    0.572    0.064 /Users/project/src/gekkoapps/common/locations/base.py:97(render_view)
     9    0.000    0.000    0.242    0.027 /Users/project/src/gekkoapps/common/locations/base.py:173(get_data_from_view)
     9    0.000    0.000    0.242    0.027 /Users/project/src/gekkoapps/common/views/base.py:136(get_data)
     9    0.000    0.000    0.239    0.027 /Users/project/src/gekkoapps/gosearch/v1/web/view/compat.py:14(create_location_data)
     9    0.000    0.000    0.238    0.026 /Users/project/src/gekkoapps/gosearch/v1/web/view/produce.py:518(get_data)
     9    0.000    0.000    0.220    0.024 /Users/project/src/gekkoapps/common/locations/base.py:183(render_json)
     9    0.000    0.000    0.220    0.024 /Users/project/src/gekko/template/helpers.py:148(do_json)
     9    0.013    0.001    0.220    0.024 /Users/project/src/.env3/lib/python3.7/site-packages/simplejson/encoder.py:371(encode)
  3626    0.030    0.000    0.214    0.000 /Users/project/src/gekko/net/resolver.py:185(resolve)
     27    0.000    0.000    0.209    0.008 /Users/project/src/gekkoapps/common/views/serp/v1/creator.py:23(create)
```

# Профилирование памяти

```
pip install memory_profiler
```

```
# run.py
from memory_profiler import profile

@profile
def some_func():
    lst1 = []
    lst2 = "1" * 100000
```

```
python -m memory_profiler run.py
```

## **top/atop**

**top** - консольная команда, которая выводит список работающих в системе процессов и информацию о них.

PID - идентификатор процесса

USER - пользователь, под которым запущен процесс

VIRT - объем виртуальной памяти, занимаемой процессом

RES - текущее использование RAM

%CPU - процент доступного времени процессора

**atop** - продвинутый интерактивный полноэкранный монитор производительности, написанный для Linux.

```
atop -r /var/log/atop/atop_<date> [-b hh:mm]
```

## iostat/iotop

**iotop** - утилита, выводящая данные по использованию жесткого диска.

- `iotop -o` (активные процессы)
- `iotop -o -a` (собрать статистику за время)

**iostat** - утилита, предназначенная для мониторинга использования дисковых разделов.

`iostat -d -t -p sda -x`

-с вывести отчет по CPU

-d вывести отчет по использованию диска

-t интервал, за который усредняются значения

-x вывести расширенную статистику

## Достойны упоминания: pdb, dis, inspect, psutil

- `python3 -m pdb script.py`
- `# script.py`  
`def some_func():`  
    `lst1 = []`  
    `import pdb; pdb.set_trace()`  
    `lst2 = "1" * 100000`  
`python script.py`

## **Домашнее задание #9**

- Сравнение скорости работы объектов с обычными атрибутами, слотами и викрефами
- Выполнить профилирование по вызовам/памяти
- Декоратор для профилирования
- flake8 + pylint перед сдачей

# Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

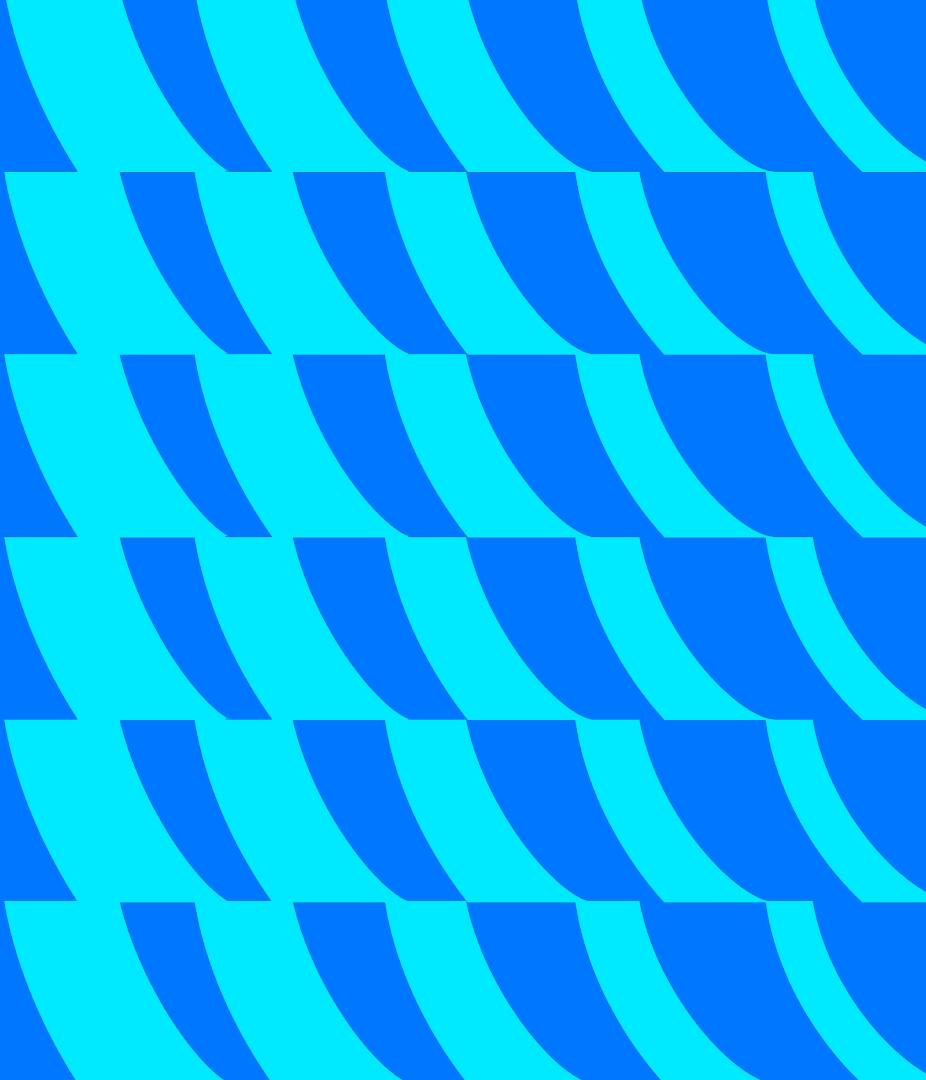
Below the schedule, a blog post titled 'Углубленный Python' is shown. It has 57 readers and 2 topics. There are buttons for 'Подписаться' (Subscribe) and 'Создать топик' (Create topic). A search bar and a 'Найти' (Find) button are also present.

On the right side, there's a sidebar for 'Прямой эфир' (Live Stream) with a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

Спасибо за  
внимание



образование



# Углубленный Python

Лекция 10

Логирование, отладка, форматирование

Кандауров Геннадий



образование

# Напоминание отметиться на портале

+ ОСТАВИТЬ ОТЗЫВ

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

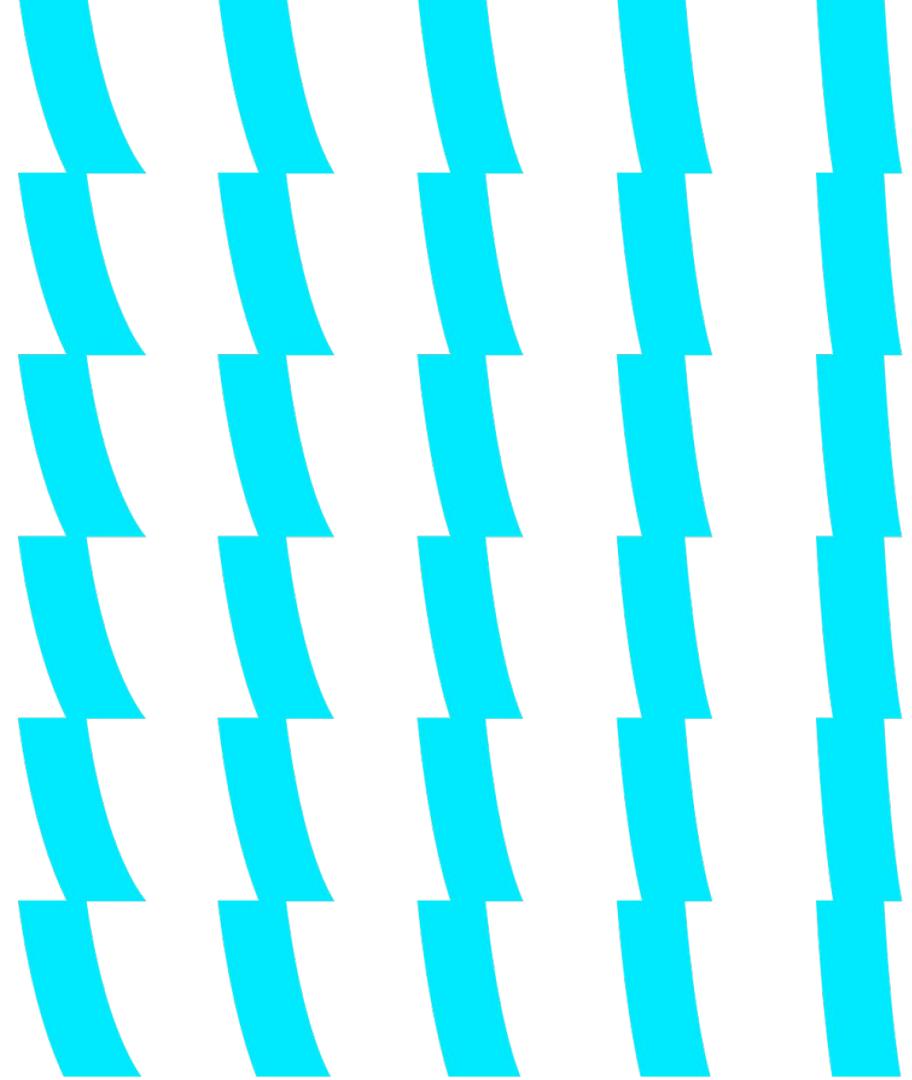
Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

Below the schedule, there's a blog post titled 'Углубленный Python' (Advanced Python) with the subtitle 'Blog for materials on the "Advanced Python" course'. It has 57 readers and 2 topics. A search bar and a 'Найти' (Find) button are below the post. To the right, there's a sidebar for 'Прямой эфир' (Live Stream) and a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

## Содержание занятия

1. Логирование
2. Отладка
3. Форматирование кода

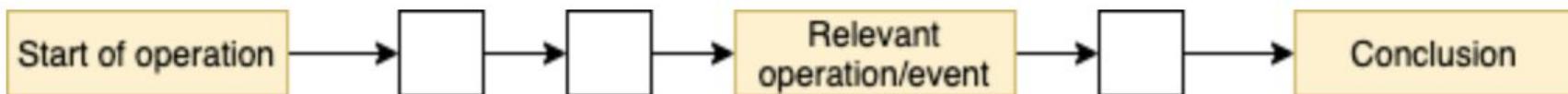
# Логирование



# Логирование

Логи должны быть:

- Наглядными
- Контекстными
- Реактивными



## Логирование: уровни

Level	Numeric value
CRITICAL	50
ERROR	40
WARNING	30
INFO	20
DEBUG	10
NOTSET	0

# Логирование

- Логгеры
- Хендлеры
- Фильтры
- Форматтеры

# Логирование

The basic classes defined by the module, together with their functions, are listed below.

- Loggers expose the interface that application code directly uses.
- Handlers send the log records (created by loggers) to the appropriate destination.
- Filters provide a finer grained facility for determining which log records to output.
- Formatters specify the layout of log records in the final output.

# Логирование

```
import logging
logging.basicConfig(
    filename="example.log",
    level=logging.DEBUG,
)
logging.debug("This message should go to the log file")
logging.info("So should this")
logging.warning("And this, too")
```

```
$ cat example.log
DEBUG:root:This message should go to the log file
INFO:root:So should this
WARNING:root:And this, too
```

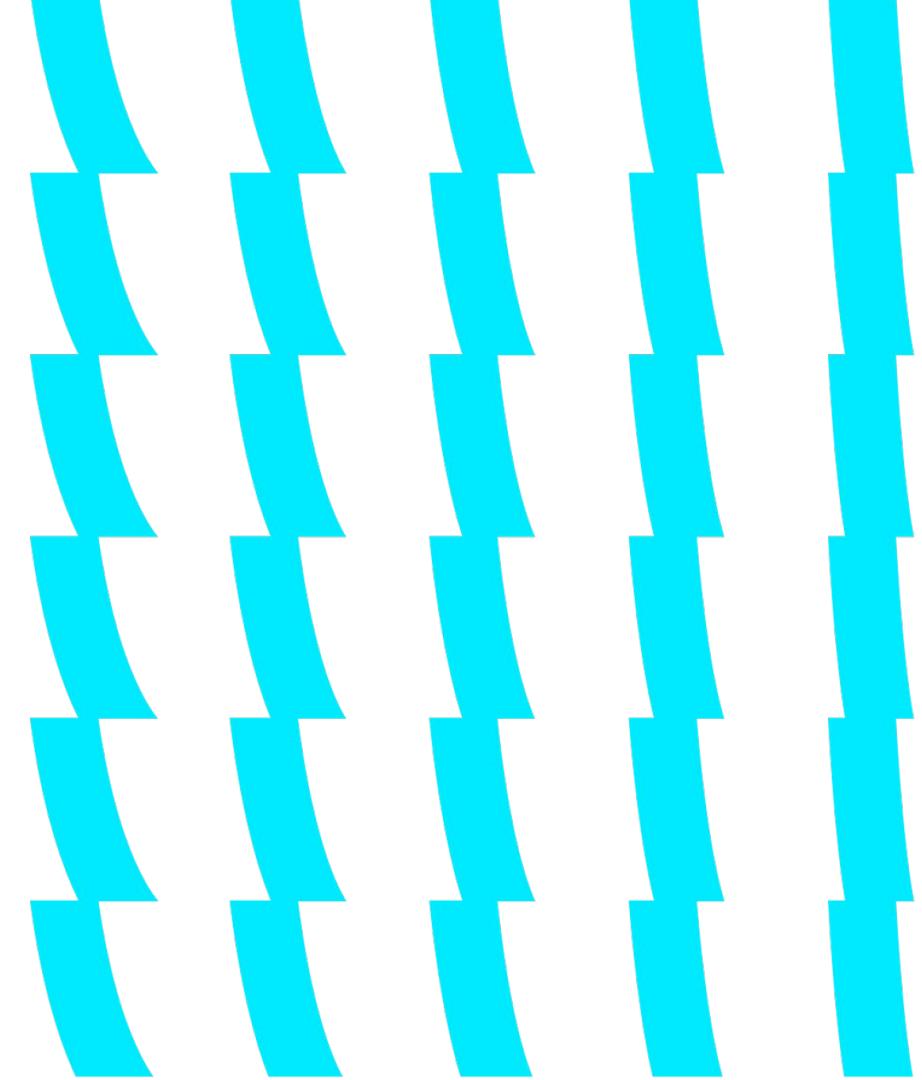
# Логирование

```
import logging
```

```
logger = logging.getLogger(__name__)
```

```
def do_action(data):  
    logger.info("run do_action with %s", data)
```

# Отладка



# dis

```
import dis
```

```
def do_action(data):  
    logger.info("run do_action with %s", data)
```

```
dis.dis(do_action)
```

```
dis.show_code(do_action)
```

# pdb

```
import pdb
```

```
def do_action(data):
    pdb.set_trace()
    logger.info("run do_action with %s", data)
```

```
do_action(99)
```

```
python3 -m pdb some_script.py # для всей программы целиком
```

# inspect

```
import inspect

def do_action(data):
    logger.info("run do_action with %s", data)

inspect.getmembers(do_action)
inspect.getmodule(do_action)
inspect.currentframe()
inspect.stack()
inspect.isgenerator(obj)
inspect.iscoroutine(obj)
inspect.isbuiltin(obj)
inspect.signature(do_action)
```

# Форматирование

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

## **Домашнее задание #10**

- Добавить логирование в LRUCache
- По аргументу командной строки  
дополнительно логировать в stdout с  
отдельным форматом
- flake8 + pylint перед сдачей

# Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции

The screenshot shows the VK Education website interface. At the top, there's a navigation bar with links like 'Блоги' (Blogs), 'Люди' (People), 'Программа' (Program), 'Вакансии' (Jobs), and 'Расписание' (Schedule). A yellow banner at the top right says 'Открыт приём заявок!' (Applications are open!). Below the banner, a weekly schedule is displayed:

Чт, 8 сентября	пт, 9 сентября	сб, 10 сентября	вс, 11 сентября	пон, 12 сентября
Нет занятий	18:00 Углублённый Py... Введение в Python, основные понятия, тестирование Г. Кандауров	Нет занятий	Нет занятий	Нет занятий

Below the schedule, a blog post titled 'Углубленный Python' is shown. It has 57 readers and 2 topics. There are buttons for 'Подписаться' (Subscribe) and 'Создать топик' (Create topic). A search bar and a 'Найти' (Find) button are also present.

On the right side, there's a sidebar for 'Прямой эфир' (Live Stream) with a list of recent comments from users like Геннадий Кандауров, Екатерина Черкасова, and Дарья Вовченко.

Спасибо за  
внимание



образование

