



Бэкенд-разработка на Python. Лекция N°1. Интенсив по Python

Алена Елизарова





Не забудьте
отметиться
на портале



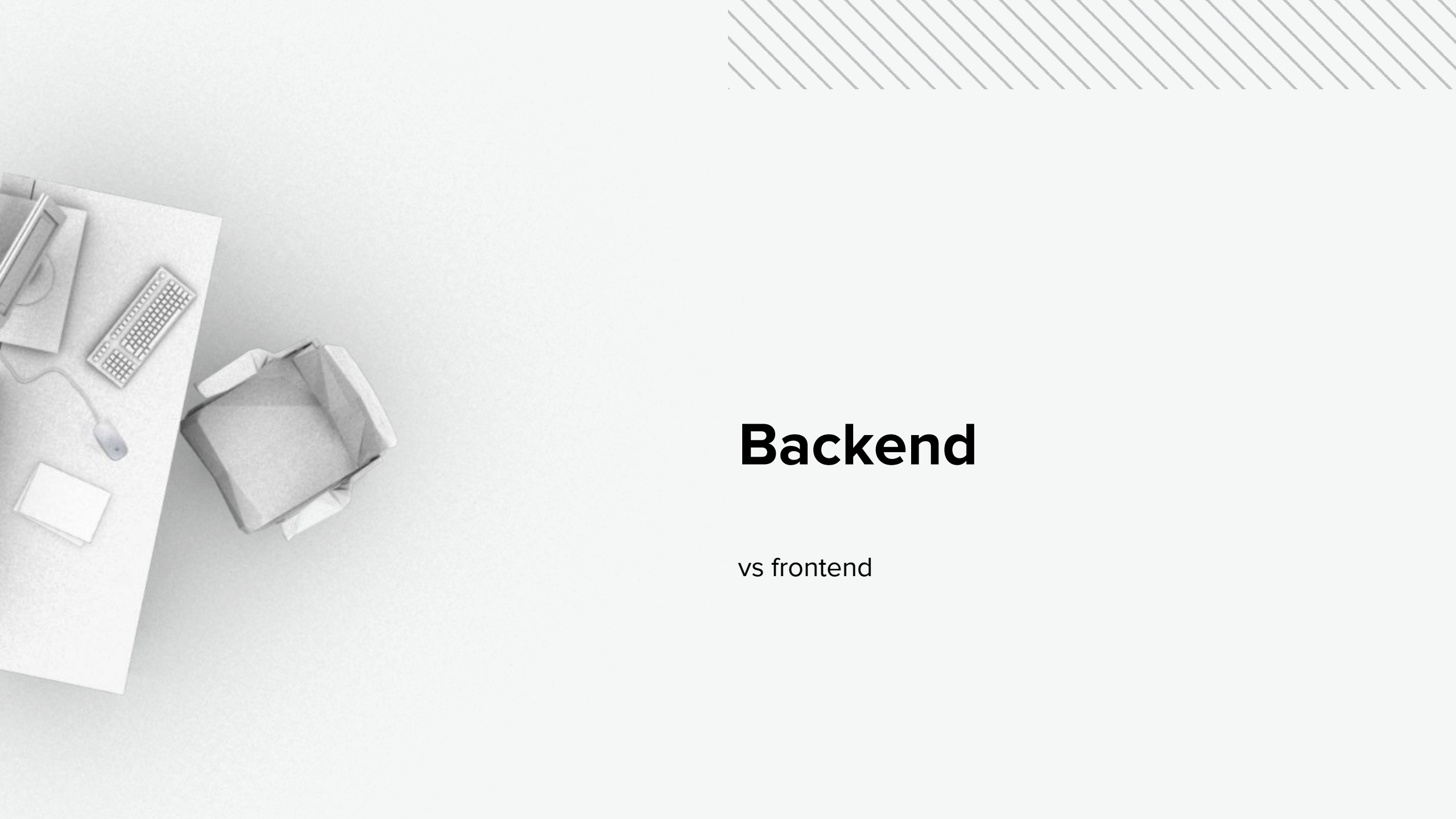
План курса

1. Интенсив Python
2. Типы и структуры данных, метаклассы, память
3. Основы http, клиент-сервер
4. Application server
5. Работа с БД
6. Django ORM
7. Разработка API
8. Авторизация в веб-приложениях
9. Потоки и процессы. Асинхронное программирование
10. Реал-тайм сообщения. Очереди и задачи.
11. Поисковые движки.
12. Микросервисная архитектура. Контейнеризация. CI/CD
13. Безопасность в веб-приложениях



План занятия

1. Обзор технологий backend
2. Поговорим о Python
3. Вспомним синтаксис языка
4. Работа с файлами
5. Юнит-тестирование
6. Стил ь языка
7. Несколько интерактивов



Backend

vs frontend



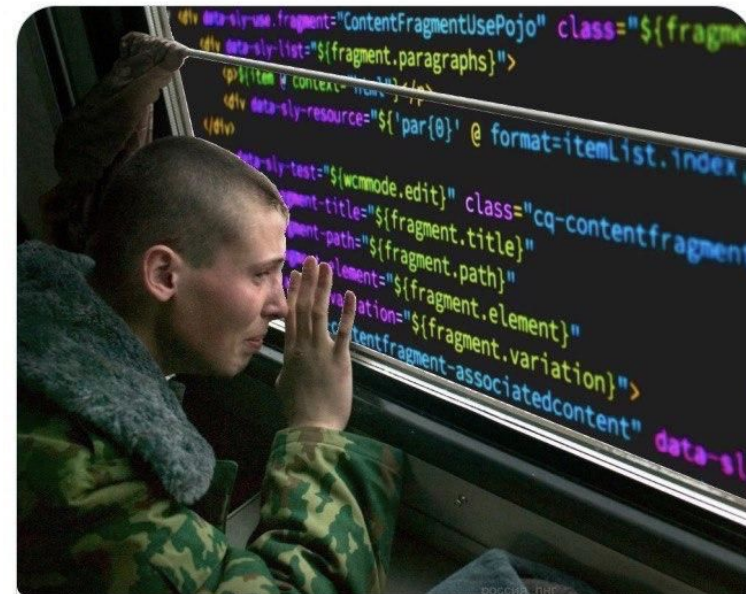
Roadmap бэкенд-разработчика



<https://roadmap.sh/backend>



Me showing my friend how his 1000 line C++ code can be written in 10 lines in Python



не хочу на фронт

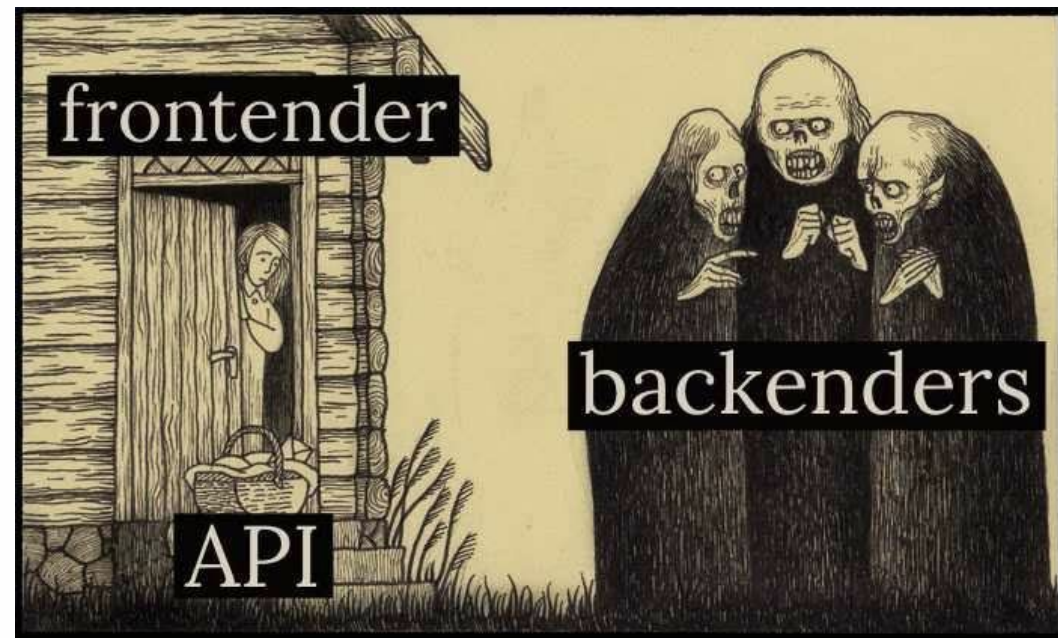
12:44

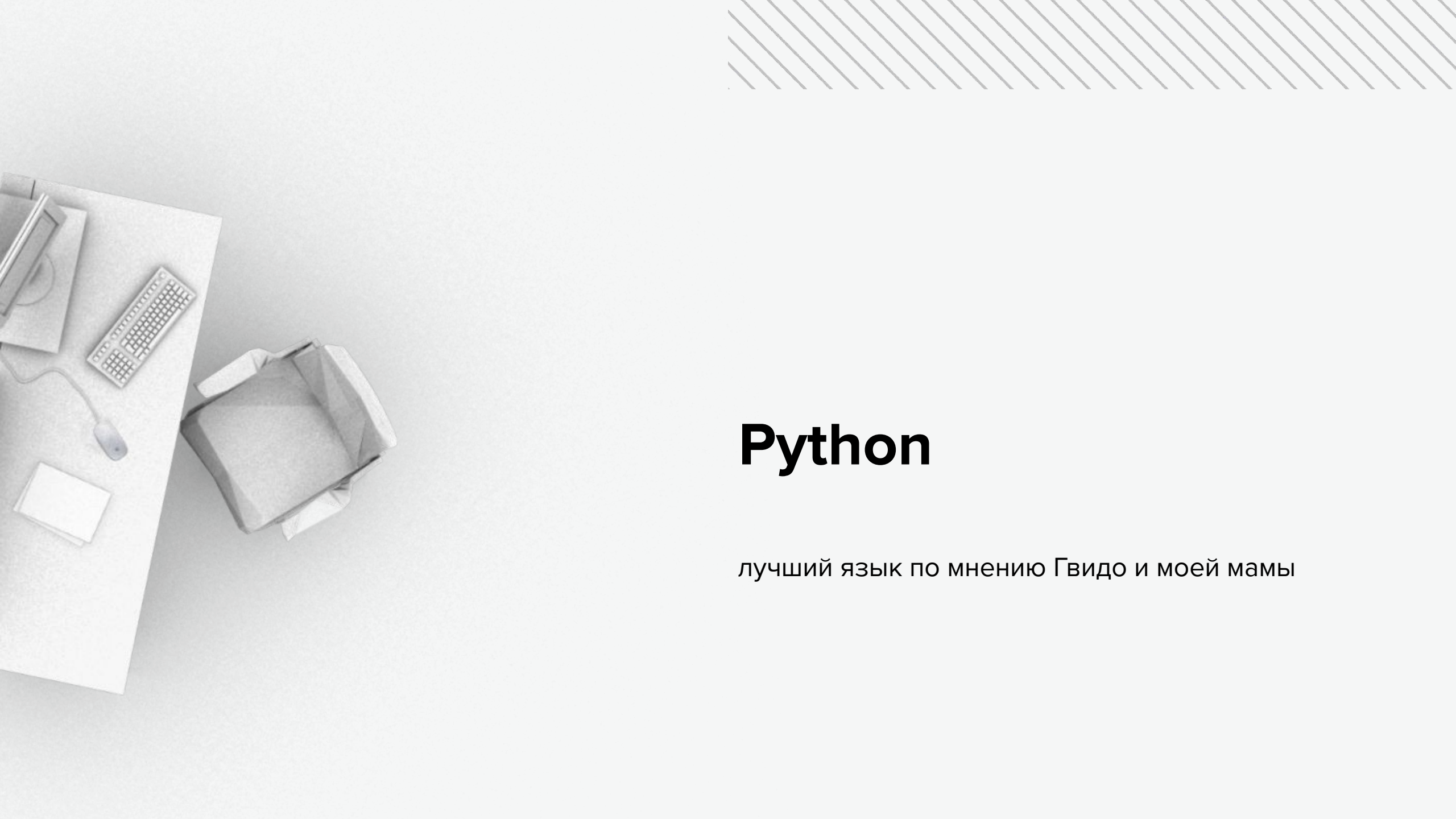
BACKEND

FRONTEND



Два запроса тормозит - сдемлайте один





Python

лучший язык по мнению Гвидо и моей мамы

Python

Интерпретируемый язык с динамической (утиной) типизацией и автоматической сборкой мусора.

Python - название спецификации языка

Реализации:

CPython

IronPython (DotNet)

PyPy

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

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



Гвидо Ван Россум
(создатель языка Python)

Переменные

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

```
# допустимые символы: буквы, цифры, _
# НО: начинается с буквы или _
```

```
# valid:
num = 1
_num = 1
__num = 1
num35num35 = 1
first_num = 1
```

```
# invalid:
1num = 1
```

Базовые типы и конструкции

Целые числа (int)

```
num = 42
num = 42_000_000 # начиная с python 3.6
```

Вещественные числа (float)

```
num = 3.14
```

Комплексные числа (complex)

```
num = 14 + 1j # num.real, num.imag - для доступа
              # до реальной и мнимой частей
```

Операции

Стандартные и %, **, //

Базовые типы и конструкции

Логический тип (bool) - подтип целого числа.

`a, b = b, a` # классический swap

Логические выражения **ленивы**.

Строки (str), сырые строки.

Форматирование строк - 4 способа.

Операторы **in**, **for ... in**, **while**, **if .. elif .. else** (также аналог тернарного оператора), **pass**, **break**, **continue**

Коллекции

Списки

```
a = list(); b = []
```

<https://pythonworld.ru/tipy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html>

Сортировка

```
my_sorted_list = sorted(my_list)  
my_sorted_list.sort()
```

Кортежи

```
a = (); b = tuple()
```

Словари

```
a = {}; b = dict()
```



Коллекции

Множества (также frozenset)


```
a = set(); b = {1, 2, 3}
```

List/dict comprehension

```
a = [i ** 2 for i in range(5)]  
b = {i: i // 2 for i in range(50)}
```



Функции, lambda




```
def multiply(a, b):  
    return a * b
```

Функции, lambda

```
list(filter(lambda x: x%2, [10, 111, 102, 213, 314, 515]))
```



Замыкания



Замыкание — функция, в которой все свободные переменные привязаны к тому значению, которое определено в рамках области видимости данной функции.

По факту это означает, что функции более низкого порядка имеют доступ к переменным функции более высокого порядка.

Декораторы

Функция, которая принимает функцию и возвращает функцию.

```
def decorator(func):  
    def wrapper():  
        print('Before func')  
        func()  
        print('After func')  
    return wrapper  
  
@decorator  
def my_func():  
    print('Hi, i am the main here')
```

Декораторы с параметрами

```
def decorator(decorator_arg):
    print('I am here only to make a decorator')
    def my_decorator(func):
        print('I am the actual decorator')
        def wrapper(func_arg):
            print('I am the wrapper and I will return the result of the func')
            return func(func_arg)
        return wrapper
    return my_decorator

@decorator(42)
def func(arg):
    print('That is the answer')
```




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
```

Итераторы

```
class MyIterator:
    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 = MyIterator(3)
print(next(iter_obj))
```

Генераторы

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

```
def lets_count(num):  
    while num > 0:  
        yield num  
        num -= 1
```

```
qwe = lets_count(10)  
next(qwe)
```

Контекстные менеджеры

```
with open('file.txt', 'w') as f:  
    f.write('hello')
```

```
class MyResource:  
    def __init__(self, name):  
        print('Resource created: {}'.format(name))  
        self.name = name  
  
    def show_name(self):  
        return self.name  
  
    def close(self):  
        print('Resource closed')
```


Контекстные менеджеры

```
class MyManager:
    def __init__(self, name):
        self.resource = MyResource(name)

    def __enter__(self):
        return self.resource

    def __exit__(self, type, value, traceback):
        self.resource.close()
```

```
with MyManager('SomeTitle') as r:
    print(r.show_name())
```

Классы

```
isinstance(3, int) # проверка на принадлежность классу
```

```
# объявляем пустой класс
class Human(object):
    pass
```

```
print(dir(Human)) # методы
```

```
class Planet(object):
    count = 0 # атрибут класса

    def __init__(self, name):
        self.name = name # атрибут экземпляра

    def __str__(self):
        return self.name
```

```
planet = Planet('Earth') # создаем экземпляр класса
```

Исключения

```
my_list = ['1', '3', 'zero']

try:
    for i in my_list:
        int(i)
except ValueError:
    print('It is not a number')
except TypeError:
    print('Can it be me?')
else:
    print('All right, good job')
finally:
    print('Okay, I am done')
```



Модуль collections



`defaultdict, OrderedDict, Counter, namedtuple`

Модули и пакеты

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
```

Интерактив

```
1.  
a = b = 0  
a += 1  
  
print(a, b)
```

```
2.  
x = y = []  
x.append(1)  
  
print(x, y)
```

Интерактив

3.
a = [1, 2, 3, 4]
a.append(a)

print(a[4][4][4])

4.
12 * 5
print(_)



Интерактив

5.
`Some = type("Some", (object,), {"x": "hello"})`

6.1.
`a,b,*c = (1,2,3,4,5,6,7,8)`

6.2.
`a,*b,c = (1,2,3,4,5,6,7,8)`

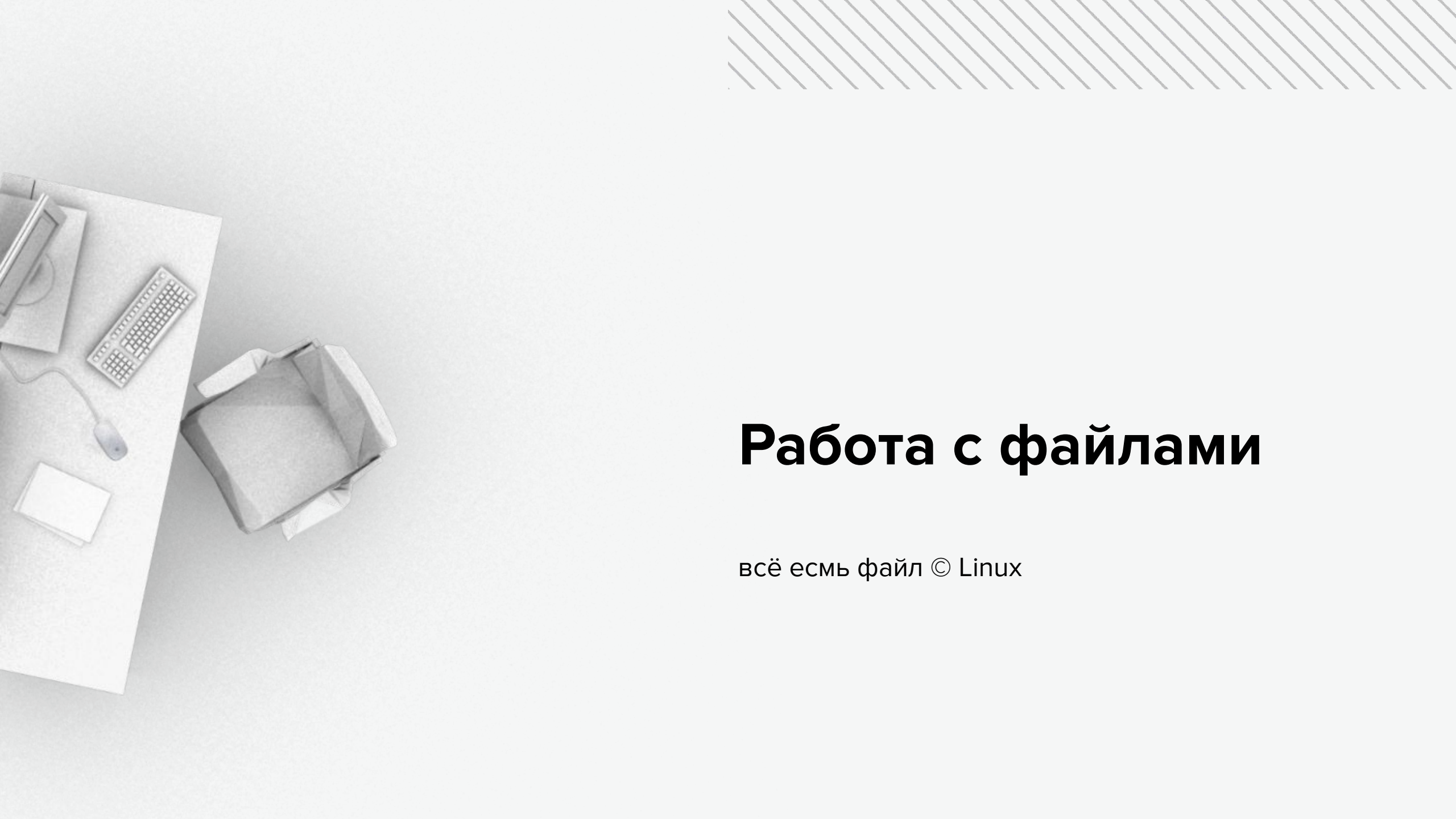
Интерактив

7.
`l = ["spam", "ham", "eggs"]
list(enumerate(l))`

8.
`'hello'[::-1]`

9.
`x = [1,2,3]
y = x[:]`

10.
`import __hello__`



Работа с файлами

всё есмь файл © Linux



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.



Модуль os

Bash vs. Python?

<https://stackoverflow.com/questions/2424921/python-vs-bash-in-which-kind-of-tasks-each-one-outruns-the-other-performance-w>

Обзор методов

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

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('test.txt')  
data = f.read()
```


```
with open('test.txt') as f:  
    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.

Методы файлового объекта



```
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)
```


Высокоуровневые библиотеки для обработки файлов

shutil

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

скопировать дерево, переместить/переименовать дерево, удалить дерево

glob

```
import glob
glob.glob('examples/*.xml')
['examples\\feed-broken.xml', 'examples\\feed-ns0.xml',
'examples\\feed.xml']
```

Wildcard	Matches	Example
*	any characters	*.txt matches all files with the txt extension
?	any one character	??? matches files with 3 characters long
[]	any character listed in the brackets	[ABC]* matches files starting with A,B or C
[..]	any character in the range listed in brackets	[A..Z]* matches files starting with capital letters
[!]	any character listed in the brackets	[!ABC]* matches files that do not start with A,B or C

Высокоуровневые библиотеки для обработки файлов

tempfile

создание временных файлов и каталогов

filecmp, difflib

сравнение файлов

filecmp.cmpfiles()

mimetypes

https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_MIME-%D1%82%D0%B8%D0%BF%D0%BE%D0%B2

работа с mime-типами

mimetypes.guess_type()

Архивы

gzip (.gz)

tarfile (.tar, .tar.gz (а также .tgz) и .tar.bz2)

zipfile (.zip)



Работа с аудио

aifc

поддержка формата AIFF (Audio Interchange File Format – формат файлов для обмена аудиоданными)

wave

возможность для работы с файлами .wav (несжатыми)

audioop

манипуляция некоторыми разновидностями аудиоданных

sndhdr

определить тип аудиоданных, хранящихся в файле, и некоторые характеристики этих данных, например, частота дискретизации

Работа с изображениями

```
from PIL import Image  
img = Image.open("some.jpg")
```

```
img.size  
.format  
.show()  
.crop(...)  
.transpose(...)
```



Популярные форматы данных

CSV (англ. Comma-Separated Values - значения, разделенные запятыми);

JSON (англ. JavaScript Object Notation) - текстовый формат обмена данными, основанный на JavaScript;

XML (англ. eXtensible Markup Language - расширяемый язык разметки);

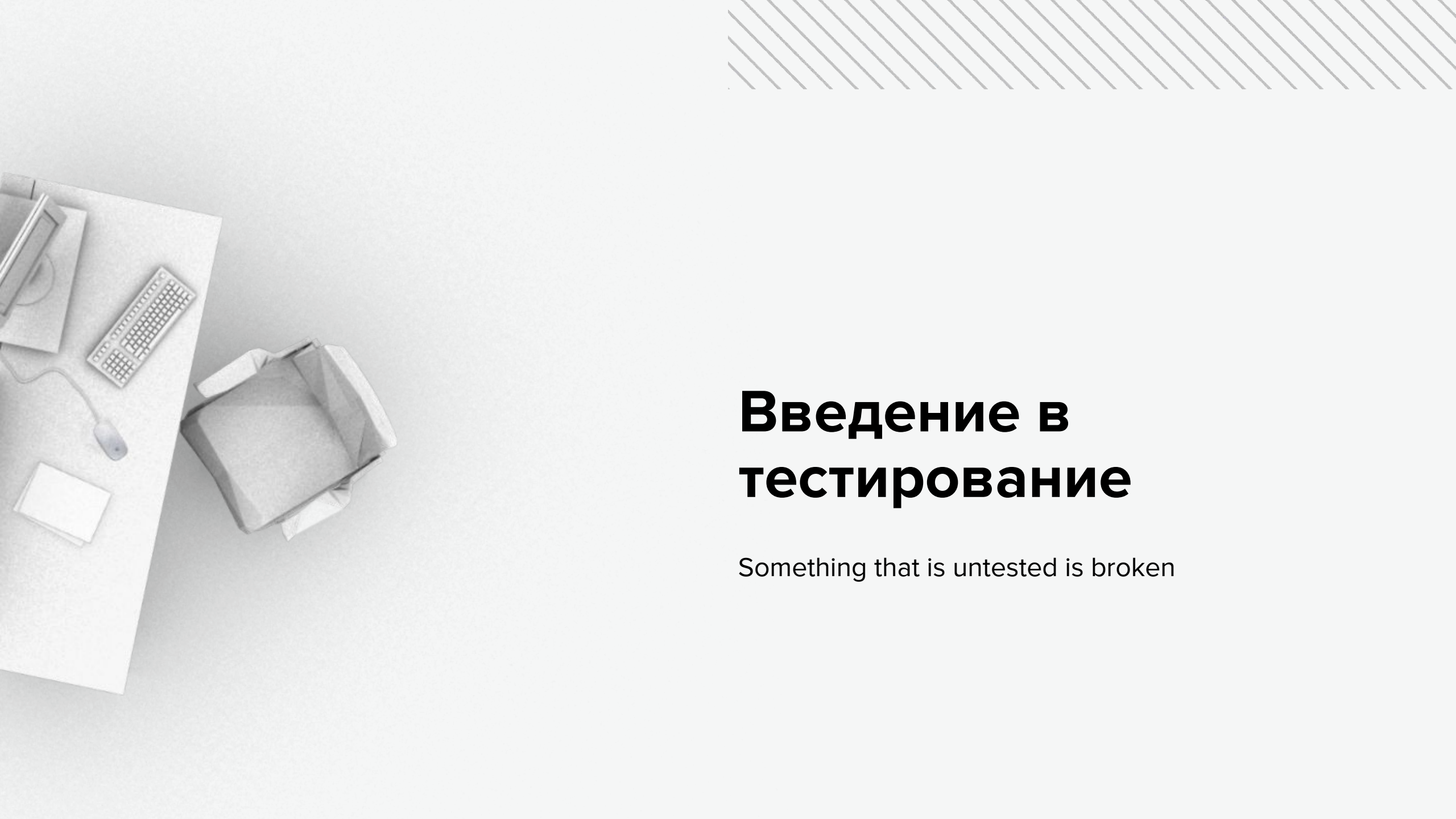
YAML (англ. YAML Ain't Markup Language - «YAML - Не язык разметки»);

CSV

```
import csv

f_name = "my_file.csv"
# список покупок
shop_list = {"киви": [2, 100], "томат": [3, 250], "тыква": [1, 35]}

# Запись в файл
with open(f_name, "w", encoding="utf-8", newline="") as f:
    writer = csv.writer(f, quoting=csv.QUOTE_ALL)
    writer.writerow(["Название", "Вес", "Цена/кг."]) # заголовки
    for name, values in sorted(shop_list.items()):
        writer.writerow([name, *values])
    writer.writerow(["огурец", "4", "170"]) # добавим
```



Введение в тестирование

Something that is untested is broken




Виды тестирования

- Unit-тестирование
 - Функциональное тестирование
 - Интеграционное тестирование
 - Нагрузочное тестирование
-
- Client side (selenium)
 - Server side

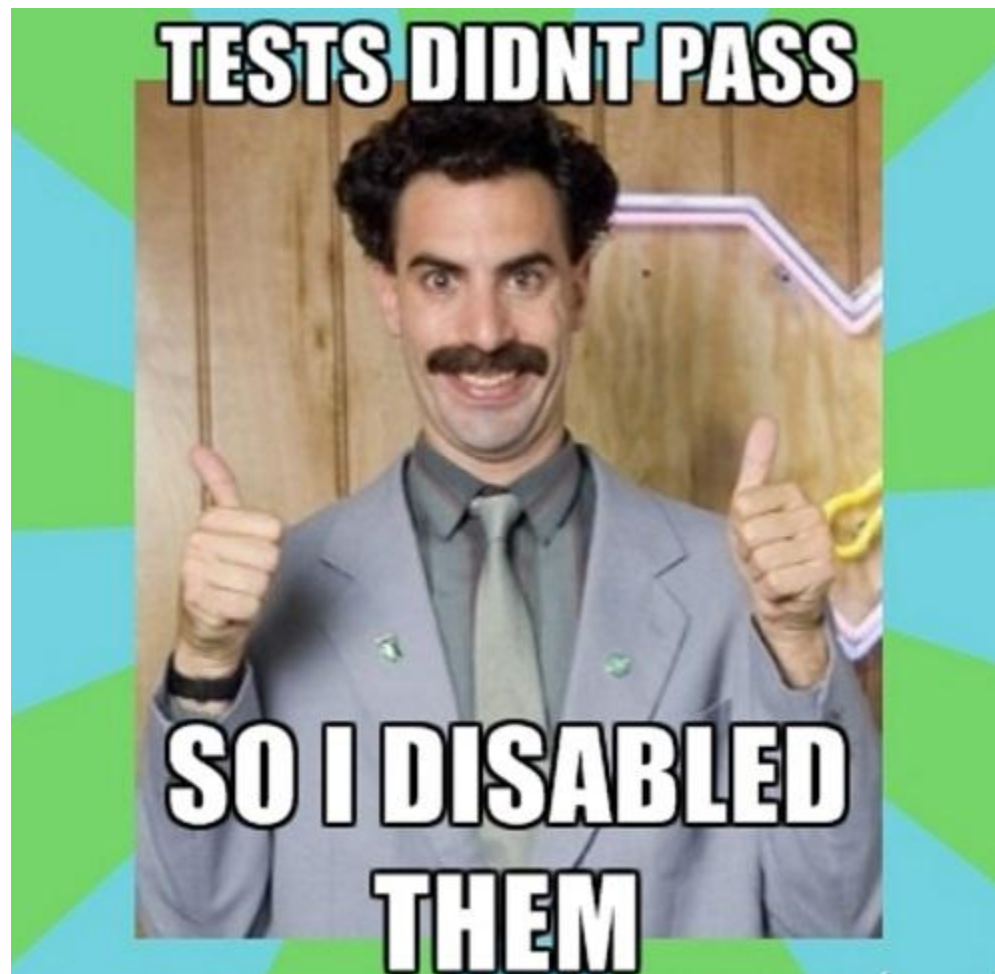
TDD - test driven development - техника разработки ПО, основывается на повторении коротких циклов разработки: пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода



Цели тестирования


- 
- Проверка правильности реализации
 - Проверка обработки внештатных ситуаций и граничных условий
 - Минимизация последствий

Цели тестирования





Инструменты для тестирования в Python

- 
- doctest
 - pytest
 - hypothesis
 - unittest

doctest

то, что мы не будем использовать

```
'''
Module showing how doctests can be included with source code
Each '>>>' line is run as if in a python shell, and counts as a test.
The next line, if not '>>>' is the expected output of the previous line.
If anything doesn't match exactly (including trailing spaces), the test fails.
'''
```

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

pytest

```
class TestClass(object):
    def test_one(self):
        x = 'this'
        assert 'h' in x

    def test_two(self):
        x = 'hello'
        assert hasattr(x, 'check')
```

Ключевое слово `assert`

Используется для проверки истинности утверждения.
Если проверка не прошла, возбуждается исключение `AssertionError`.

Рекомендуется использовать только для проверки внутреннего состояния программы — ситуаций, которые не должны происходить, которые нельзя обработать, или это не имеет смысла (обычно это является указанием на то, что код программы содержит ошибку). Также может использоваться для документирования ожиданий (например, входных параметров или результата).

```
passed = False  
assert passed, 'Not passed' # возбуждается исключение AssertionError
```

unittest

```
import unittest
from main import get_result

class SomeTest(unittest.TestCase):
    def setUp(self):
        self.data = {'students': 5}

    def test_result(self):
        result = get_result(self.data)
        self.assertEqual(result, 5)

    def tearDown(self):
        print('I am done')

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


unittest. Расширенный набор проверок assert

```
assertEqual(a, b)
assertNotEqual(a, b)
assertTrue(x)
assertFalse(x)
assertIs(a, b)
assertIsNot(a, b)
assertIsNone(x)
assertIn(a, b)
assertIsInstance(a, b)
assertLessEqual(a, b)
assertListEqual(a, b)
assertDictEqual(a, b)
assertRaises(exc, fun, *args, **kwargs)
assertJSONEqual(a, b)
```

mock

Mock - подменяет объекты (функции, классы) на так называемые mock-объекты, заглушки.

```
class TestUserSubscription(TestCase):  
    @patch('users.views.get_subscription_status')  
    def test_subscription(self, get_subscription_status_mock):  
        get_subscription_status_mock.return_value = True  
        ...
```

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

пример

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

freezegun

```
from freezegun import freeze_time
from datetime import datetime
from unittest import TestCase

class TestWithDatetime(TestCase):

    def setUp(self):
        self.dt = datetime(
            year=2018, month=8, day=1, hour=12, minute=00
        )

    def test_something(self):
        meeting_id = 1
        data = {'data': 'some_data'}
        with freeze_time(self.dt):
            update_meeting(meeting_id, data)
```

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`

Почувствуй себя интерпретатором

```
from django.test import TestCase
from unittest import mock

class SomeTest(TestCase):

    @mock.patch("confroom.notification_helpers.send_templated_email")
    @mock.patch("confroom.notification_helpers.send_sms")
    def test_create(self, send_email_mock, send_sms_mock):
        """
        дано: 2 пользователя, у обоих стоит уведомления по смс,
        у одного стоит уведомлять по email
        """
        self.create_meeting()

        self.assertEqual(send_sms_mock.call_count, 2)
        self.assertEqual(send_email_mock.call_count, 1)
```

Почувствуй себя интерпретатором

```
from django.test import TestCase

class SomeTest(TestCase):

    def test_error1(self):
        with self.assertRaises(SyntaxError):
            int(b)

    def test_error2(self):
        with self.assertRaises(TypeError):
            int('b')

    def test_error3(self):
        with self.assertRaises(KeyError):
            {}['key']
```



PER8

каждый код хочет быть красивым



PEP8. Линтеры

Код читается намного больше раз, чем пишется. Гвидо

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

<https://www.pylint.org/>

<http://flake8.pycqa.org/>

<https://habr.com/ru/company/oleg-bunin/blog/433480/>

PEP8. Основные тезисы

Функции верхнего уровня и определения классов отделяются **двумя пустыми строками**.

Определения методов внутри класса разделяются **одной пустой строкой**.

Импорты помещаются **в начале** файла.

Импорты должны быть сгруппированы в следующем порядке:

- импорты из **стандартной** библиотеки
- импорты **сторонних** библиотек
- импорты модулей **текущего** проекта



PEP8. Основные тезисы

Внутри скобок на границах **нет пробелов**

Всегда окружайте бинарные операторы **одним пробелом** с каждой стороны

Комментарий к функции нужно писать **после строки с def**.

Имена классов **ClassName**

Имена функций **this_is_function**

Имена констант **THIS_IS_CONSTANT**

Сравнения с None: **if var is None**



Домашнее задание по Лекции 1

ДЗ #1

- Реализовать игру крестики-нолики в виде класса
- Написать тесты (unittest) для игры
- Проверить корректность и стиль кода с помощью pylint или flake8



**Не забудьте
оставить отзыв на
портале**



**СПАСИБО
ЗА ВНИМАНИЕ**

