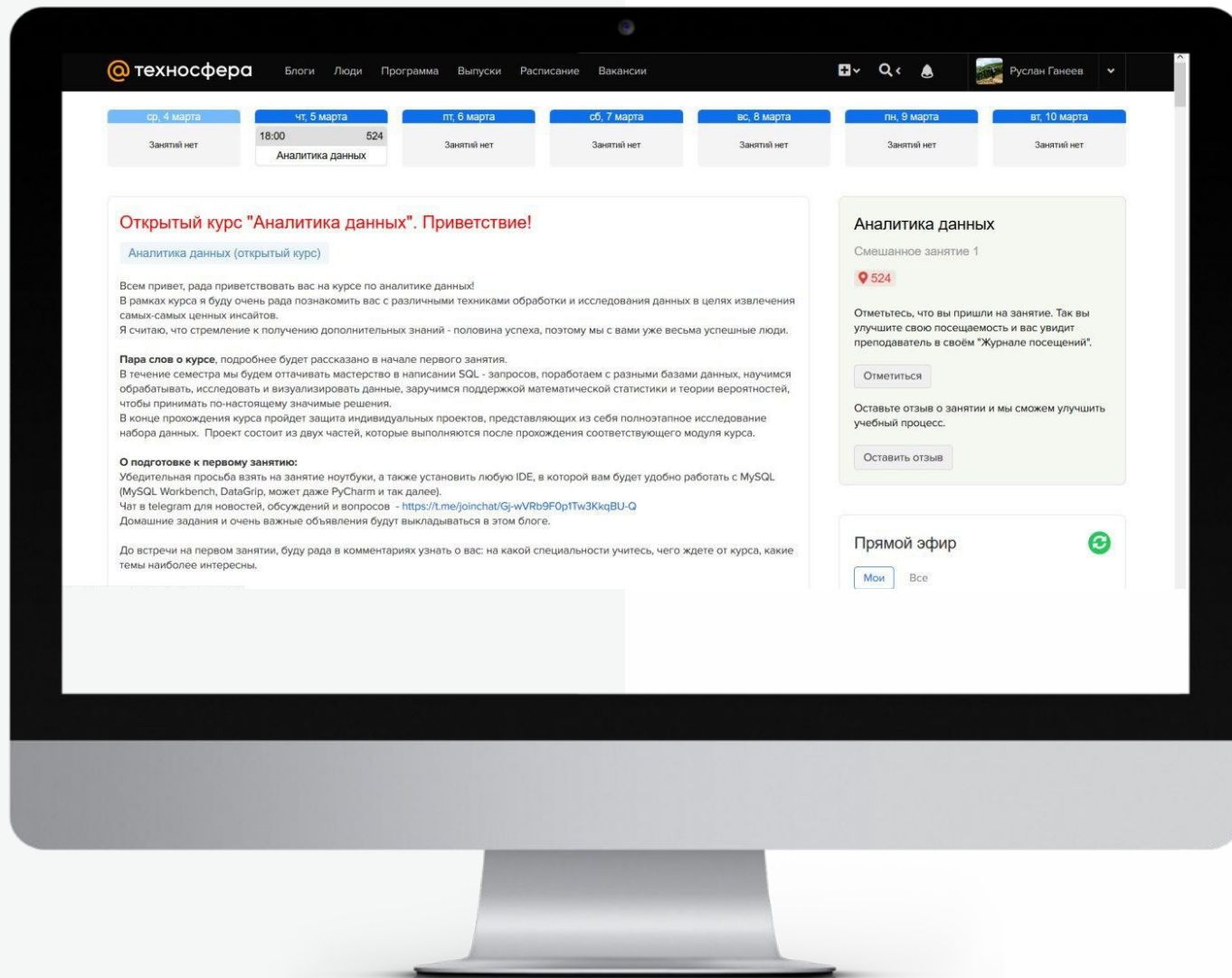




Бэкенд-разработка на Python. Лекция N°3. Метаклассы

Алена Елизарова



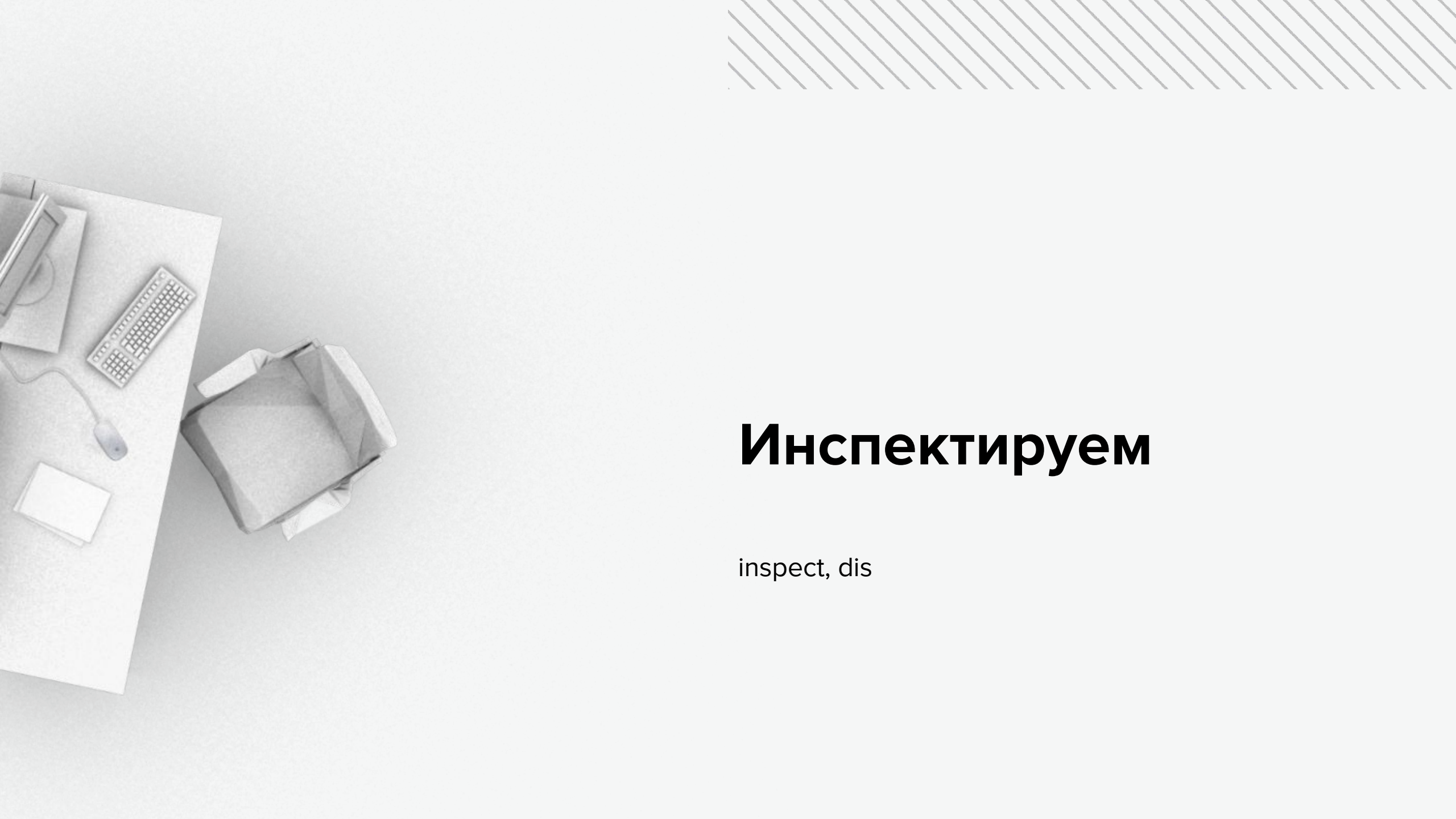


Не забудьте
отметиться
на портале



План занятия

1. Inspect, dis
2. Дескрипторы
3. Метаклассы
4. MRO
5. ABC



Инспектируем

inspect, dis

dis

```
python3 -m timeit 'pow(3,89)'
```

```
python3 -m timeit '3**89'
```

dis

```
python3 -m timeit 'pow(3,89)'  
500000 loops, best of 5: 488 nsec per loop
```


```
python3 -m timeit '3**89'  
500000 loops, best of 5: 417 nsec per loop
```

<https://github.com/python/cpython/blob/master/Include/opcode.h>

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



Inspect



Модуль, который предоставляет пачку полезных функций для получения информации об объектах в python



Inspect




```
inspect.getmembers
```

```
# Return all the members of an object in a list of (name, value) pairs sorted  
by name.
```




Inspect. Source code



`inspect.getdoc`

`inspect.getfile`

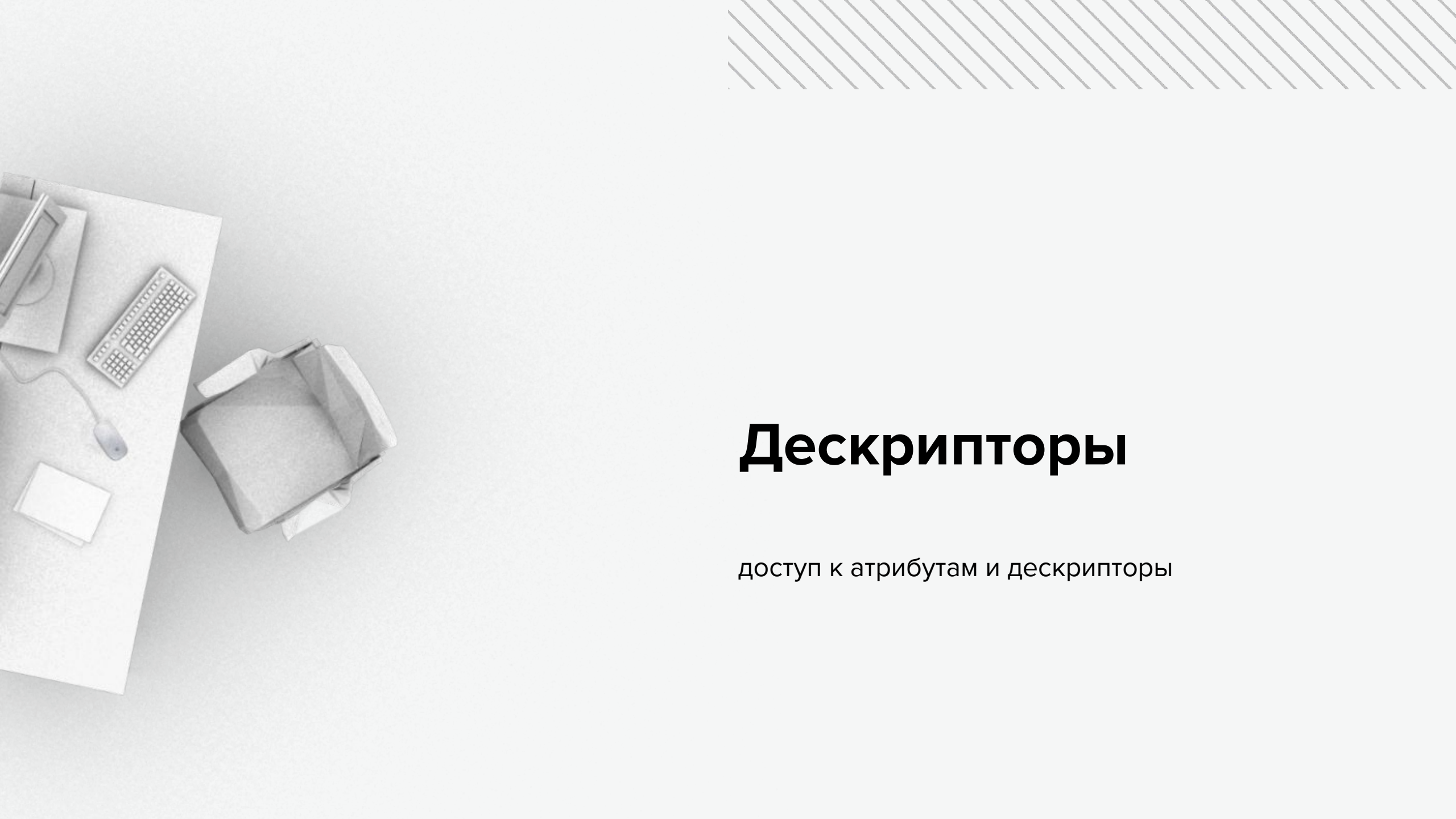
`inspect.getmodule`

`inspect.getsourcefile`

`inspect.getsource`

Inspect. Signature

```
>>> def foo(a, *, b:int, **kwargs):  
...     pass  
  
>>> sig = signature(foo)  
  
>>> str(sig)  
'(a, *, b:int, **kwargs)'  
  
>>> str(sig.parameters['b'])  
'b:int'  
  
>>> sig.parameters['b'].annotation  
<class 'int'>
```



Дескрипторы

доступ к атрибутам и дескрипторы

Доступ к атрибутам

Рассмотрим подробнее атрибут `__dict__`

Чтобы найти атрибут объекта `o`, python обыскивает:

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

Дескрипторы

```
>>> 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

“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 метод дескриптор.”

Раймонд Хеттингер



Дескрипторы

Если определен один из методов на предыдущем слайде - объект считает дескриптором.

Если объект дескриптора определяет `__get__`, `__set__` - он считает data дескриптором.

Если объект дескриптора определяет `__get__` - он считает non-data дескриптор.

Дескрипторы

```
class MyDescriptor:
    def __set__(self, obj, val):
    ...
    def __get__(self, obj, objtype):
    ...
    def __delete__(self, obj):
    ...

class MyClass:
    field1 = MyDescriptor()
    field2 = MyDescriptor()
```

Дескрипторы

```
class StaticMethod(object):  
    "Эмуляция PyStaticMethod_Type() в Objects/funcobject.c"  
  
    def __init__(self, f):  
        self.f = f  
  
    def __get__(self, obj, objtype=None):  
        return self.f
```

Дескрипторы

```
from sqlalchemy import Column, Integer, String

class User(Base):
    id = Column(Integer, primary_key=True)
    name = Column(String)
```

Дескрипторы

```
class Order:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

    def total(self):
        return self.price * self.quantity

apple_order = Order('apple', 1, 10)
apple_order.total()
```

Дескрипторы

```
class Order:
    price = NonNegative('price')
    quantity = NonNegative('quantity')
    def __init__(self, name, price, quantity):
        self._name = name
        self.price = price
        self.quantity = quantity
    def total(self):
        return self.price * self.quantity
apple_order = Order('apple', 1, 10)
apple_order.total()
# 10
apple_order.price = -10
# ValueError: Cannot be negative
apple_order.quantity = -10
```



Дескрипторы

Методы доступа к атрибутам (yet another магия)

Методы `__getattr__()`, `__setattr__()`, `__delattr__()` и `__getattribute__()`.

В отличие от дескрипторов их следует определять для объекта, содержащего атрибуты и вызываются они при доступе к любому атрибуту этого объекта.



Магические методы. Кастомизация объектов



`object.__new__(cls[, ...])` – создает новый объект класса, статический метод по преданию.

После создание объекта вызывается (уже у объекта) метод `__init__`. Он ничего не должен возвращать, иначе будет `TypeError`

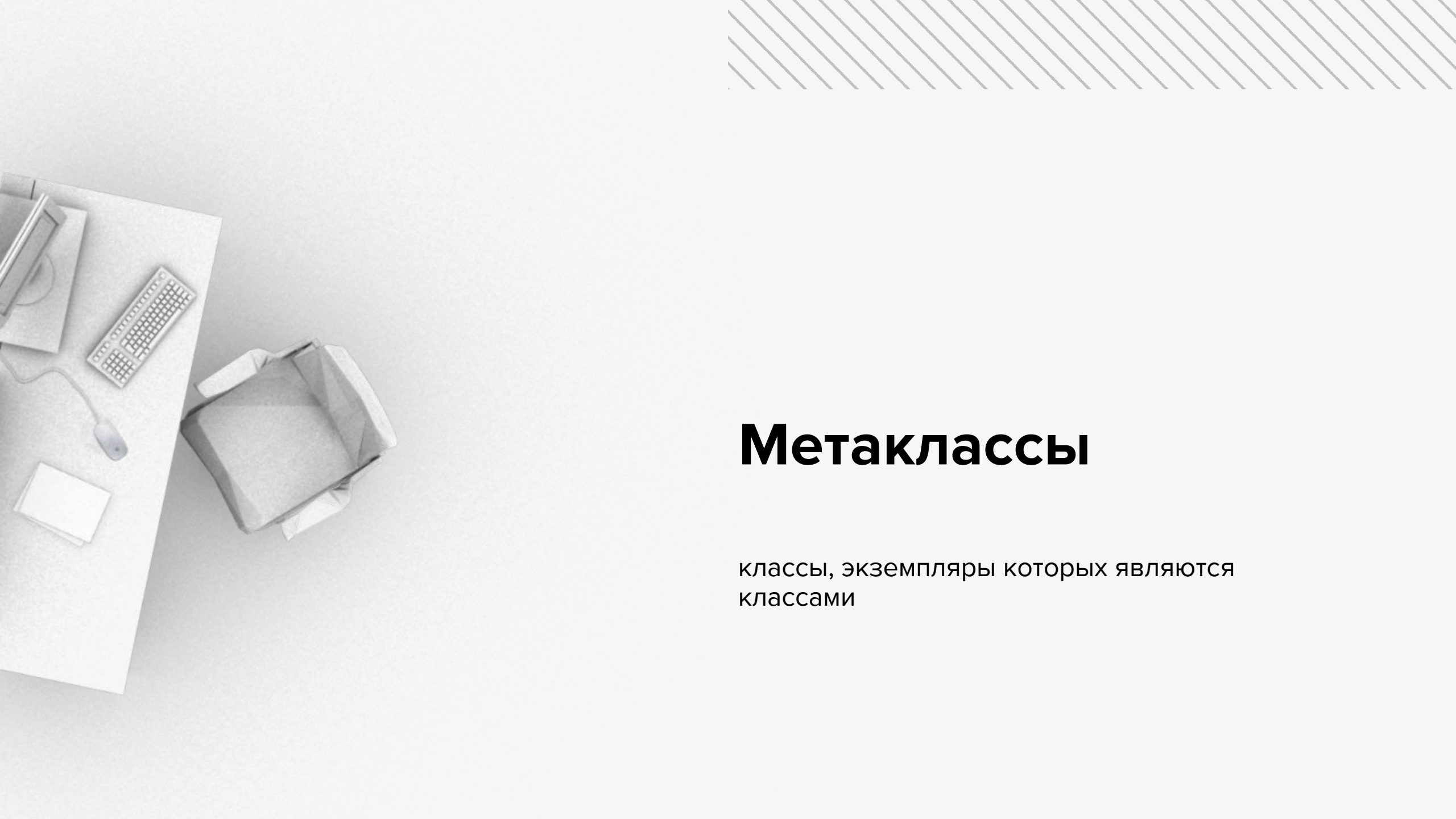
Кастомизация объектов. Подумать

```
>>> class Foo:
...     pass
...
>>> x = Foo()
>>> type(x)
<class '__main__.Foo'>
>>> type(Foo)
???
>>> type(type)
???
```

Кастомизация

```
class Singleton(object):
    _instance = None

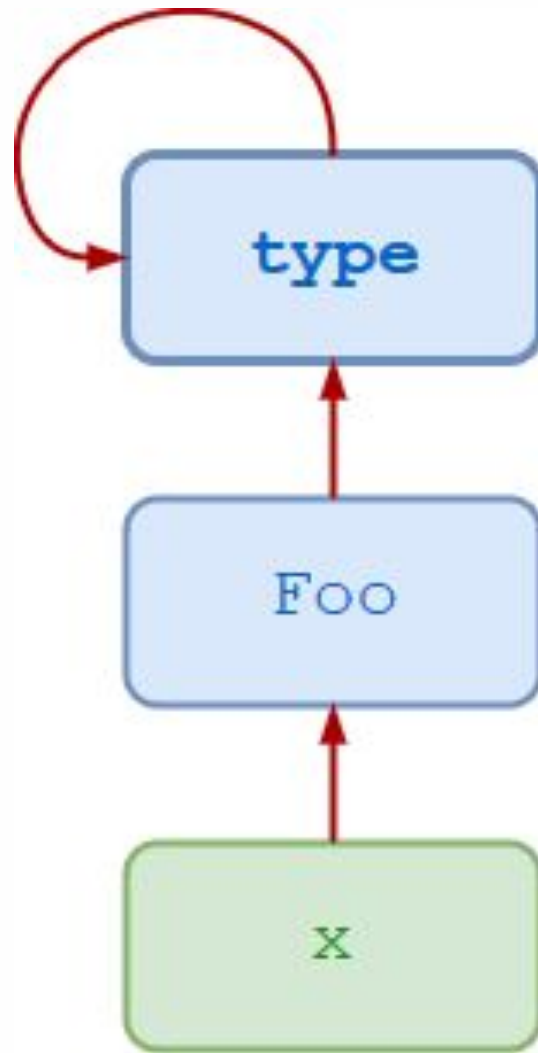
    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super().__new__(cls, *args, **kwargs)
        return cls._instance
```

Метаклассы

классы, экземпляры которых являются классами

type



#026

Метаклассы

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', (), {})
```

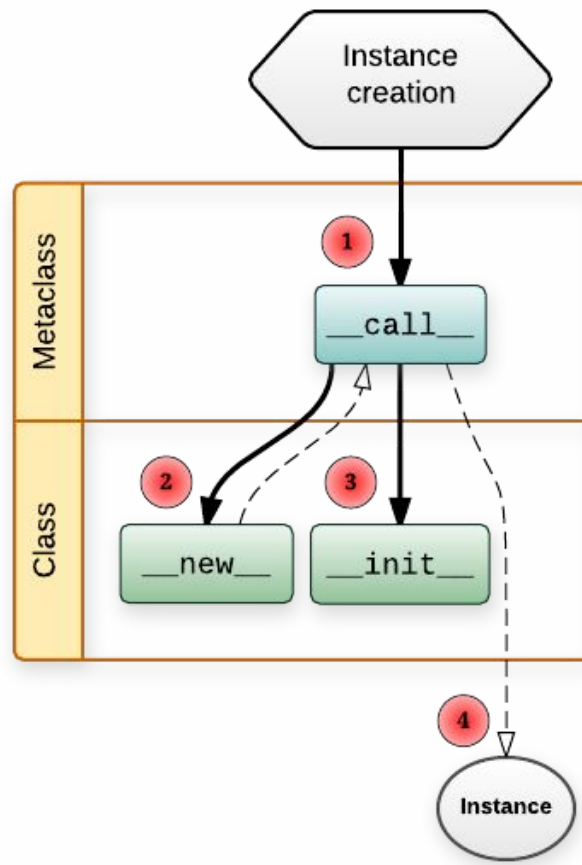
Метаклассы

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

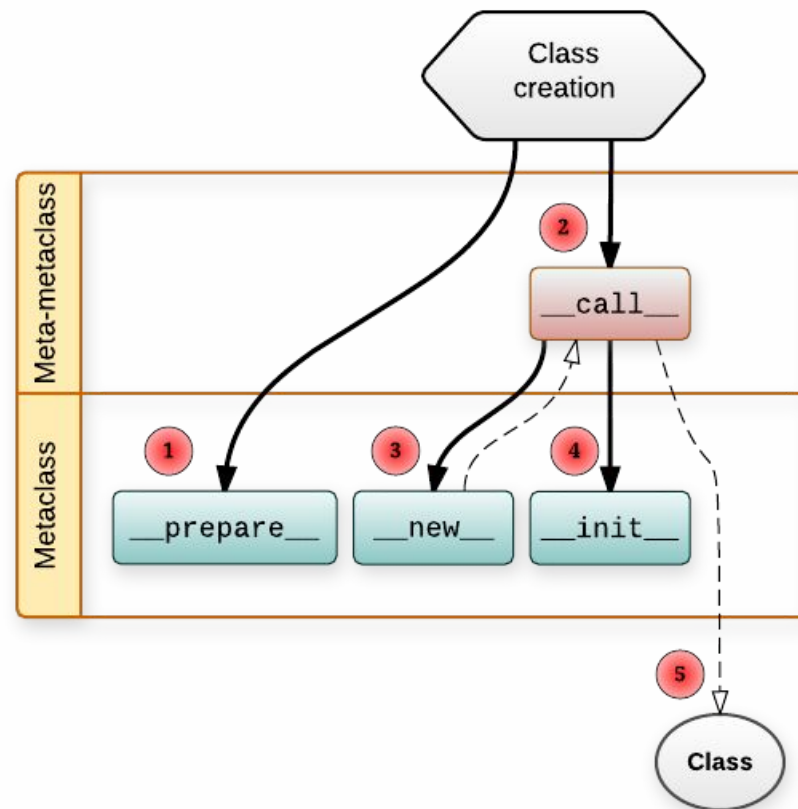
>>> class Bar(Foo):
...     attr = 100
...
>>> x = Bar()
>>> x.attr
100
>>> x.__class__.__bases__
(<class '__main__.Foo'>,)

```

Создание объекта



Создание класса





Как создается класс

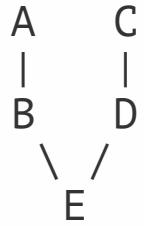
- определяются базовые классы
- определяется метакласс
- подготавливается namespace класса (`__prepare__`)
- выполняется тело класса
- создается класс (`__new__`, `__init__`)

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.

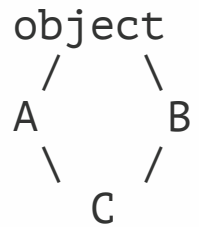
```
__mro__  
.mro()
```

MRO до Python 2.2



E, B, A, D и C


MRO c Python 2.2



C, A, object, B



Проблема «ромбовидной структуры»



Если у нас есть классы А и В, от которых наследуется класс С, то при поиске метода по старому алгоритму получается, что если метод не определён в классах С и А он будет извлечён из object, даже если он определён в В.

Упорядоченный список классов, в которых будет производиться поиск метода слева направо будем называть **линеаризацией** класса.

Решение

Линеаризация должна быть **монотонной**.

Если в линеаризации некого класса C класс A следует за классом B (она имеет вид $[C, \dots, B, \dots, A]$) и для любого его потомка D класс B будет следовать за A в его линеаризации (она будет иметь вид $[D, \dots, C, \dots, B, \dots, A]$), то линеаризация будет **монотонной**.

$L[A] = [A, \text{object}]$; $L[B] = [B, \text{object}]$

$L[C] = [C, A, \text{object}, B] \Rightarrow L[C]$ – не удовлетворяет условию

Локальный порядок старшинства

Линеаризация, которые удовлетворяют свойству монотонности:

$L[C] = [C, A, B, \text{object}]$

$L[C] = [C, B, A, \text{object}]$

Какой выбрать?

Определяется **порядок локального старшинства** — это свойство, которое требует соблюдения в линеаризации класса-потомка того же порядка следования классов-родителей, что и в его объявлении.

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'>]
```




ABC

добавляем абстракции



ABC



Модуль, который позволяет определять абстрактные базовые классы (abstract base classes).

ABC example. Hashable

```
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
```

ABC example

```
>>>from abc import *
>>>class C(metaclass = ABCMeta):
...     @abstractmethod
...     def absMethod(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 absMethod
```

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



ORM

оборачиваем SQL



Пишем ORM

ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

Технология ORM позволяет проектировать работу с данными в терминах классов, а не таблиц данных. Она позволяет преобразовывать классы в данные, пригодные для хранения в базе данных, причем схему преобразования определяет сам разработчик. Кроме того, ORM предоставляет простой API-интерфейс для CRUD-операций над данными. Благодаря технологии ORM нет необходимости писать SQL-код для взаимодействия с локальной базой данных.



Ссылки

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

<https://docs.python.org/3/reference/datamodel.html>

<https://www.python.org/download/releases/2.3/mro/>

<https://habr.com/ru/post/62203/>

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

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

Домашнее задание

- Написать ORM для реляционной базы (MySQL, PostgreSQL)

https://en.wikipedia.org/wiki/Active_record_pattern

https://en.wikipedia.org/wiki/Data_mapper_pattern

<https://medium.com/oceanize-geeks/the-active-record-and-data-mappers-of-orm-pattern-eefb8262b7bb>



Домашнее задание

Нужно реализовать CRUD:

- Метод создания `.create`
- Метод извлечения данных `.all + .get`
- Метод обновления `.update()`
- Метод удаления `.delete()`



Домашнее задание по Лекции 3

ДЗ #1

#048

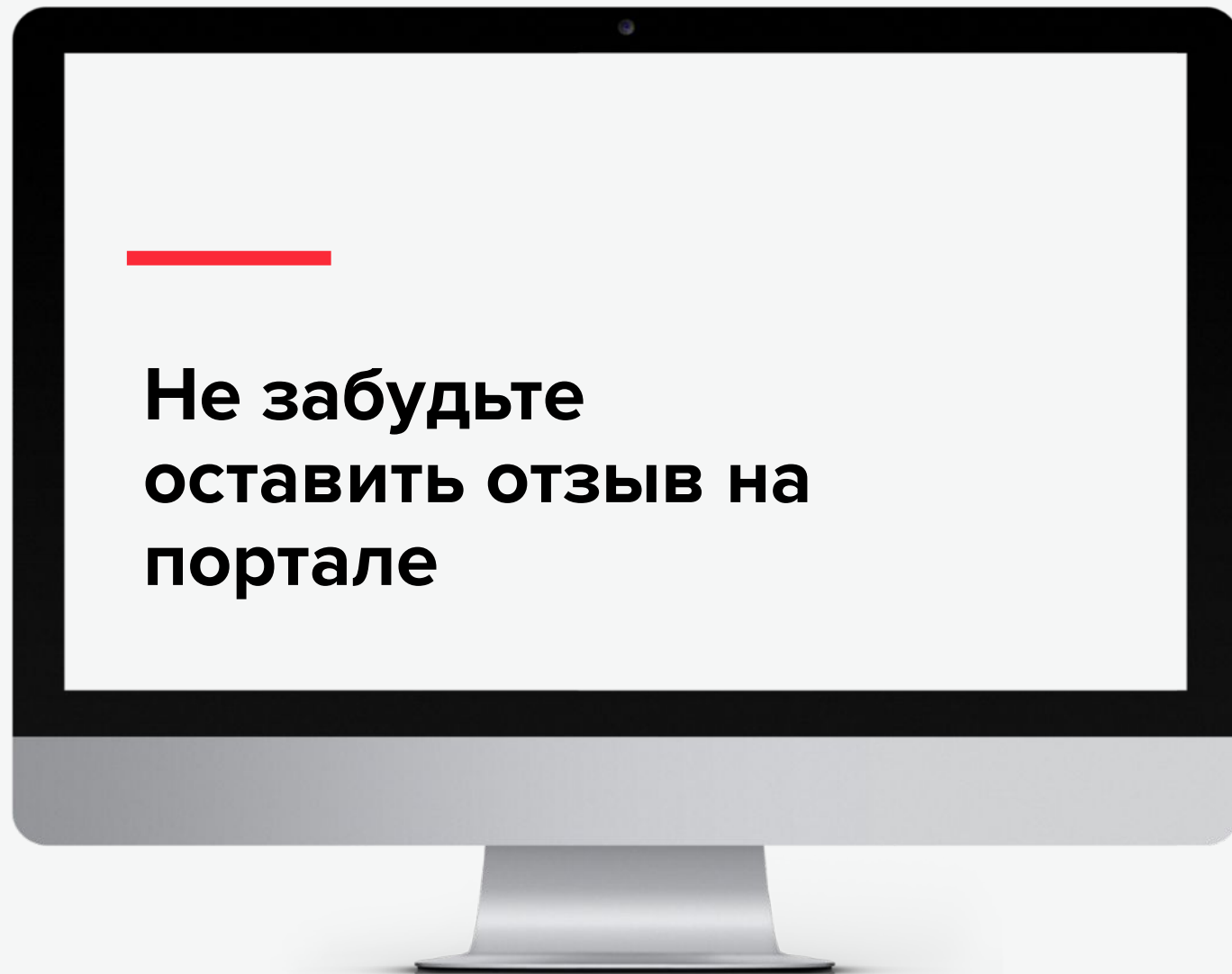
65

Баллов
за задание

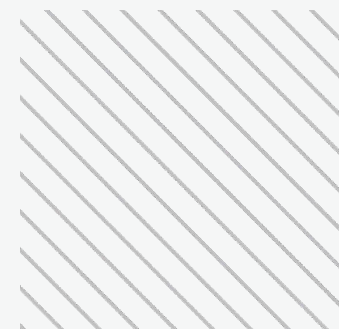
09.11.20

Срок
сдачи

Написать свой аналог ORM



**Не забудьте
оставить отзыв на
портале**



**СПАСИБО
ЗА ВНИМАНИЕ**

