OOP

@pvavilin

7 мая 2022 г.

Outline

Класс

```
class Animal(object):
    ...

class Cat(Animal):
    ...
```

Яндекс Практикум

Объект

```
a_cat = Cat(type="египетский")
```

Яндекс Практикум

Объект

```
a_cat = Cat(type="кроличий")
```

Яндекс Практикум

Метод

```
class Cat(Animal):
    ...
    def sleep(self, period):
        self.activity = None
        time.sleep(period)
```

Атрибут

```
class Cat(Animal):
    tail_state = 'puffed up'
    ...
```

Атрибут класса

Так мы говорим, что у всех кошек на Земле хвост задран

```
class Cat(Animal):
    tail_state = 'puffed up'
    ...
```

А так только у какой-то конкретной кошки (у конкретного экземпляра объекта класса *Cat*)

```
class Cat(Animal):
    def __init__(self):
        self.tail_state = 'puffed up'
    ...
```

Яндекс Практикум

self

self это указать на конкретный экземпляр объекта класса.

Яндекс Практикум

self

```
class Cat(Animal):
    def meow(self):
        print('Meow')

cat_1 = Cat()
cat_2 = Cat()
cat_3 = Cat()
cat_1.meow()
# Кто сказал "мяу"?
```

Инициализация

```
class Cat(Animal):
    def __init__(self,
                 age,
                 breed,
                 color_schema,
                 tail_state,
                 is_domestic=False,
                 family=None):
        self.age = age
        self.breed = breed
        self.color_schema = color_schema
        self.tail_state = tail_state,
        self.is_domestic = is_domestic
        self.family = family
```

Яндекс Практикум

Инициализация

```
def take_cat(self, family):
    if is_domestic:
        raise Exception(
            'Это домашняя кошка!'
            ' Её нельзя забрать!'
        )
    self.is_domestic = True
    self.family = family
```

@dataclass

документация

```
from dataclasses import dataclass

@dataclass
class Cat(Animal):
    age: int
    breed: str
    color_schema: int
    tail_state: int
    is_domestic: bool = False
    family: object = None
```

Яндекс Практикум

@dataclass

документация

```
class Cat(Animal):
  def __init__(
    self, age: int, breed: str,
    color_schema: int, tail_state: int,
    is_domestic: bool=False,
    family: object=None
  ):
    self.age = age
    self.breed = breed
    self.color_schema = color_schema
    self.tail_state = tail_state,
    self.is_domestic = is_domestic
    self.family = family
```

Яндекс Практикум

@dataclass

```
@dataclasses.dataclass(
    *, init=True, repr=True, eq=True,
    order=False, unsafe_hash=False,
    frozen=False
)
```

Наследование

```
class Animal:
    def say(self):
        raise NotImplementedError()

class Cat(Animal):
    def say(self):
        print('meow!')

class Dog(Animal):
    def barking(self):
        # что произойдёт?
        return self.say()
```

Наследование по-простому

Наследование это всего лишь порядок, по которому будет идти поиск атрибутов и методов.

```
# class A(object):
class A:
    attr = 10
    def method(self):
        print("method")

class B(A):
    attr = 20
    def function(self):
        print("function")
```

Яндекс Практикум

super

super() возвращает объект родителя, чтобы мы могли запросить нужный нам метод/атрибут у родителя

```
class B(A):
    def function(self):
        super().method()
        (
            self.__class__
            .__bases__[0]
            .method(self)
        )
        print("function")
```

Яндекс Практикум

super

```
b = B()
b.function()
```

```
method
method
function
```

Полиморфизм

```
def listen_to_animal(animal: Animal):
    animal.say()

a_cat = Cat()
a_dog = Dog()
listen_to_animal(a_cat)
listen_to_animal(a_dog)
```

Инкапсуляция

- переменные и методы с одним подчёркиванием _name программисты договорились считать внутренними переменными
- переменные и методы с двойным подчёркиванием __name Python прячет особым образом (но к ним всё ещё можно получить доступ)

Инкапсуляция

```
class A:
    x = 10
    _y = 20
    __z = 30

a = A()
print(a.x)    # -> 10
print(a._y)   # -> 20
try:
    a.__z # -> ERROR!
except AttributeError as e:
    print(e)
print(a._A__z)
```

Яндекс Практикум

Статические методы

Не требуют указания текущего объекта вызова

```
class Cat:

    @staticmethod
    def say():
        print("meow")

class AnotherCat:
    pass

def say():
    print("meow")
```

Яндекс Практикум

Статические методы

```
AnotherCat.say = staticmethod(say)
cat_1 = Cat()
cat_2 = AnotherCat()
cat_1.say()
cat_2.say()
```

В Python всё есть объект

```
a = 10
print(a.bit_length())

def func(x, y):
    return x + y

f = func
print(f.__name__)
```

```
4
func
```

Дополнительная литература

- Object Oriented Programming in Python
- Основы ООП. Классы, объекты, методы
- wikibook
- Марк Лутц. «Изучаем Python»

Вопросы-ответы