Наследование в Python

- наследование классов
- множественное наследование
- вызов super()
- name mangling
- композиция vs наследование

Зачем нужно наследование классов?

- изменение поведения класса
- расширение функционала класса

Класс Pet, домашний питомец

In [ ]:
```
class Pet:
    def __init__(self, name=None):
        self.name = name
```

Наследование, класс Dog

In [ ]:
```
class Dog(Pet):
    def __init__(self, name, breed=None):
        super().__init__(name)
        self.breed = breed

    def say(self):
        return "{0}: waw".format(self.name)

>>> dog = Dog("Шарик", "Доберман")
>>> print(dog.name)
Шарик
>>> print(dog.say())
Шарик: waw
>>>
```

Множественное наследование

In [ ]:
```
import json

class ExportJSON:
    def to_json(self):
        return json.dumps({
            "name": self.name,
            "breed": self.breed
        })

class ExDog(Dog, ExportJSON):
    pass

>>> dog = ExDog("Белка", breed="Дворняжка")
>>> print(dog.to_json())
{"name": "\u0411\u0435\u043b\u043a\u0430", "breed": "\u0414\u0432\u043e\u0440\u043d\u044f\u0436\u043a\u0430"}
```

Любой класс является потомком object

In [ ]:
```
>>> issubclass(int, object)
True
>>> issubclass(Dog, object)
True
>>> issubclass(Dog, Pet)
True
>>> issubclass(Dog, int)
False
```

Объект является экземляром класса?

In [ ]:
```
>>> isinstance(dog, Dog)
True
>>> isinstance(dog, Pet)
True
>>> isinstance(dog, object)
True
```

Поиск атрибутов и методов объекта, линеаризация класса

In [ ]:
```
#      object
#     /   \
#    /     \
#  Pet    ExportJSON
#   |       /
#  Dog    /
#    \   /
#    ExDog

# Method Resolution Order
>>> ExDog.__mro__
(<class '__main__.ExDog'>, <class '__main__.Dog'>,
 <class '__main__.Pet'>, <class '__main__.ExportJSON'>,
 <class 'object'>)
>>>
```

Использование super()

In [ ]:
```
>>> ExDog.__mro__
(<class '__main__.ExDog'>, <class '__main__.Dog'>,
 <class '__main__.Pet'>, <class '__main__.ExportJSON'>,
 <class 'object'>)

class ExDog(Dog, ExportJSON):
    def __init__(self, name, breed=None):
        # вызов метода по MRO
        super().__init__(name, breed)
        # super(ExDog, self).__init__(name)

class WoolenDog(Dog, ExportJSON):
    def __init__(self, name, breed=None):
        # явное указание метода конкретного класса
        super(Dog, self).__init__(name)
        self.breed = "Шерстяная собака породы {0}".format(breed)

>>> dog = WoolenDog("Жучка", breed="Такса")
>>> print(dog.breed)
Шерстяная собака породы Такса
```

Разрешение конфликта имен, name mangling

In [ ]:
```
class Dog(Pet):
    def __init__(self, name, breed=None):
        super().__init__(name)
        self.__breed = breed

    def say(self):
        return "{0}: waw!".format(self.name)

    def get_breed(self):
        return self.__breed

class ExDog(Dog, ExportJSON):
    def get_breed(self):
        return "порода: {0} - {1}".format(self.name, self.__breed)

>>> dog = ExDog("Фокс", "Монс")
>>> dog.__dict__
{'name': 'Фокс', '_Dog__breed': 'Монс'}
>>> dog.get_breed()
```

Композиция классов или наследование?

In [ ]:
```
class Pet:
    pass

class Dog(Pet):
    pass

class ExportJSON(Pet):
    pass

class ExDog(Dog, ExportJSON)
    pass
```

Композиция VS наследование

In [ ]:
```
class ExportJSON:
    def to_json(self):
        pass

class ExportXML:
    def to_xml(self):
        pass

class ExDog(Dog, ExportJSON, ExportXML):
    pass

>>> dog = ExDog("Фокс", "монс")
>>> dog.to_xml()
>>> dog.to_json()
```

In [ ]:
```
### Композиция классов против наследования, пример буду вводить в онлайн

import json


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


class Dog(Pet):
    def __init__(self, name, breed=None):
        super().__init__(name)
        self.breed = breed

    def say(self):
        return "{0}: waw".format(self.name)


class PetExport:
    def export(self, dog):
        raise NotImplementedError


class ExportXML(PetExport):
    def export(self, dog):
        return """<xml version="1.0" encoding="utf-8">
<dog>
  <name>{0}</name>
  <breed>{1}</breed>
</dog>""".format(dog.name, dog.breed)


class ExportJSON(PetExport):
    def export(self, dog):
        return json.dumps({
            "name": dog.name,
            "breed": dog.breed,
        })


class ExDog(Dog):
    def __init__(self, name, breed=None, exporter=None):
        super().__init__(name, breed)

        self._exporter = exporter or ExportJSON()

        if not isinstance(self._exporter, PetExport):
            raise ValueError("bad export instance value", exporter)

    def export(self):
        return self._exporter.export(self)


>>> fox = ExDog("Фокс", "Монс", exporter=ExportXML())
>>> print(fox.export())
<xml version="1.0" encoding="utf-8">
<dog>
  <name>Фокс</name>
  <breed>Монс</breed>
</dog>
>>>
>>> muhtar = ExDog("Мухтар", "питбуль")
>>> print(muhtar.export())
{"name": "\u0414\u0436\u0435\u043a", "breed": "\u043c\u043e\u043f\u0441"}
```