ООП - Наследование

№ урока: 2 **Курс:** Python Essential

Средства обучения: PyCharm

Обзор, цель и назначение урока

После завершения урока обучающиеся расширят своё представление о парадигме объектно-ориентированного программирования и её реализации в языке Python, смогут понимать и использовать принципы наследования.

Изучив материал данного занятия, учащийся сможет:

- Понимать механизмы наследования и множественного наследования
- Создавать иерархии классов в программах на Python
- Понимать, что такое MRO

Содержание урока

1. Что такое наследование?
2. Что такое множественное наследование?
3. Реализация наследования в Python
4. Реализация множественного наследования в Python
5. Как работает наследование в Python, что такое MRO?
6. Что такое super?
7. Реализация в Python

Резюме

Наследование - один из основных принципов ООП. Этот механизм позволяет вам создать расширенный класс других классов.
Подкласс унаследует атрибуты и методы из родительского класса. Он так же может переопределять (**override**) методы родительского класса. Например, если подкласс не определяет свой инициализатор, он унаследует его от родительского класса по умолчанию.
Наследование в объектно-ориентированном программировании похоже на наследование в реальной жизни, когда ребенок наследует характеристики своих родителей и создает свои собственные характеристики.
Основная идея наследования в объектно-ориентированном программировании: **класс может наследовать характеристики другого класса**. Класс, который наследует другой класс, называется **дочерним** классом. Класс, который дает наследие, называется **родительским**, или **основным**.
Например, котёнок может наследовать цвет кожи и глаз своего родителя. Но так же у котёнка может появиться собственная характеристика: особое пятнышко на шерсти. А значит, наследование - это процесс "заимствования" некоторых характеристик родителя с возможностью их переопределить или добавить новые.

Что такое множественное наследование?

Кто такой пегас: конь или птица? И то и другое: тело коня, крылья птицы. То есть, если у нас есть класс коня и класс птицы нам нужно совместить два класса в одном? Это возможно с помощью **множественного наследования** — унаследование характеристик одновременно от двух классов.
Вместе с этим механизмом появляется много вопросов и проблем. Например, если в двух родительских классах есть одинаковые методы, с какого класса метод будет в дочернем классе? Чтобы дать ответ на этот вопрос, нам надо погрузиться в механизм реализации наследования в Python.

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Реализация наследования в Python

Рассмотрим пример наследования в Python:

```
# example_1.py

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

    def say_meow(self):
        print(f"{self.name}: Meow!")

class Kitty(Cat):
    pass

my_cat = Cat("Black")
my_kitty = Kitty("Gray")

my_cat.say_meow()
my_kitty.say_meow()
```

Создан класс Cat, определён инициализатор и метод say_meow. Создан класс Kitty и унаследован весь функционал их класса Cat. **Чтобы наследовать класс, нужно вписать название родительского класса внутри скобок, которые следуют за названием дочернего класса.**

Дальше создается два объекта с класса Cat и Kitty. Kitty унаследовал инициализатор из класса Cat. Поэтому мы можем инициализировать Kitty так же, как и Cat. Унаследованы так же методы: мы можем вызвать на объекте метод say_meow.
Добавим в класс Kitty собственных методов:

```
# example_2.py

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

    def say_meow(self):
        print(f"{self.name}: Meow!")

class Kitty(Cat):
    def say_meow(self):
        print(f"{self.name}: Meow from kitty!")

    def bite_a_finger(self):
        print("Bite! :)")


my_cat = Cat("Black")
my_kitty = Kitty("Gray")

my_cat.say_meow()
my_kitty.say_meow()
my_kitty.bite_a_finger()
```

1. Переопределен метод say_meow. Это значит, что даже если мы наследуемся от другого класса с таким же методом, больший приоритет имеет наш метод.
2. Добавлен метод bite_a_finger, которого нет в родительском классе.
3. Инициализатор не переопределён, поэтому класс Kitty по-прежнему использует инициализатор класса Cat.

Реализация множественного наследования в Python

Вернемся к примеру с пегасом. Чтобы попрактиковаться, рекомендую написать самостоятельно классы Bird и Horse. Класс Bird должен иметь метод fly, класс Horse - метод run. Добавим класс Pegas:

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
# example_3.py

class Bird:
    TYPE = "Bird"

    def fly(self):
        print(f"I am a {self.TYPE} and I can fly!")

class Horse:
    TYPE = "Horse"

    def run(self):
        print(f"I am a {self.TYPE} and I can run!")

class Pegas(Bird, Horse):
    pass

my_home_pegas = Pegas()
my_home_pegas.run()
my_home_pegas.fly()
```

Теперь наш пегас умеет и летать, и бегать! Нам просто нужно указать через запятую два родительских класса. Но если запустить код, мы увидим следующее:

```
# python example_3.py

I am a Bird and I can run!
I am a Bird and I can fly!
```

Мы ни разу не указали `TYPE = "Pegas"`, поэтому логично, что мы **не** видим "**I am a Pegas and I can run!**". Почему именно Bird, а не Horse - мы поговорим в теме MRO. Сейчас исправим эту неточность (попробуйте самостоятельно):

```
# example_4.py

class Bird:
    TYPE = "Bird"

    def fly(self):
        print(f"I am a {self.TYPE} and I can fly!")

class Horse:
    TYPE = "Horse"

    def run(self):
        print(f"I am a {self.TYPE} and I can run!")

class Pegas(Bird, Horse):
    TYPE = "Pegas"

my_home_pegas = Pegas()
my_home_pegas.run()
my_home_pegas.fly()
```

Мы добавили к Pegas атрибут TYPE = "Pegas" и теперь все работает корректно.

Как работает наследование в Python, что такое MRO?

MRO (Method Resolution Order) - это порядок в котором методы ищутся в основном и родительских классах. MRO использует алгоритм C3 линеаризации. С помощью этого алгоритма Python выстраивает линейную цепочку, как надо искать метод в классе.

В первую очередь Python ищет метод или атрибут в классе, к которому принадлежит объект. После, по порядку MRO, Python ищет этот метод\атрибут в родительских классах. Вернемся к самому первому примеру и рассмотрим его со стороны MRO:

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
# example_1.py

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

    def say_meow(self):
        print(f"{self.name}: Meow!")

class Kitty(Cat):
    pass

my_cat = Cat("Black")
my_kitty = Kitty("Gray")

my_cat.say_meow()
my_kitty.say_meow()
```

my_kitty.say_meow() - мы вызываем метод, которого нет у класса этого объекта. Но он есть в классе родителя, поэтому в Python цепочка поиска метода выглядит так: (Kitty, Cat). Если словами: "Сначала ищем метод в классе Kitty, если не найдешь, ищи в Cat".

Рассмотрим пример сложнее:

```
# example_4.py

class Bird:
    TYPE = "Bird"

    def fly(self):
        print(f"I am a {self.TYPE} and I can fly!")

class Horse:
    TYPE = "Horse"

    def run(self):
        print(f"I am a {self.TYPE} and I can run!")

class Pegas(Bird, Horse):
    TYPE = "Pegas"

my_home_pegas = Pegas()
my_home_pegas.run()
my_home_pegas.fly()
```

my_home_pegas.run() вызываем метод, которого нет в пегаса. Как узнать, в каком классе Python будет искать этот метод в первую очередь? Можно детально изучить алгоритм С3 линеаризации или просто посмотреть в атрибут класса __mro__:

```
print(Pegas.__mro__)# Result: (<class '__main__.Pegas'>, <class '__main__.Bird'>, <class
'__main__.Horse'>, <class 'object'>)
```

Это та самая цепочка по которой Python ищет методы и атрибуты. Вначале Python ищет метод\атрибут в классе Bird, потом в Horse. Если классы поменять местами class Pegas(Bird, Horse):, цепочка изменится: (<class '__main__.Pegas'>, <class '__main__.Horse'>, <class '__main__.Bird'>, <class 'object'>).

Что такое super?

Иногда надо не переопределить метод, а расширить его функционал. Например:

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
# example_5.py

class MathBase:
    def math_operation(self, number_1, number_2, operation):

    if operation == "-":
        return number_1 - number_2
    elif operation == "+":
        return number_1 + number_2

class MathUpgrated(MathBase):
    def math_operation(self, number_1, number_2, operation):
        if operation in ["+", "-"]:
            parent_class_object = super()
            result = parent_class_object.make_math_operation(number_1, number_2, operation)
            return result
        elif operation == "*":
            return number_1 * number_2
        elif operation == "/":
            return number_1 / number_2

my_math = MathUpgrated()

print(my_math.math_operation(4, 2, "+")) # ==> 6
print(my_math.math_operation(4, 2, "-")) # ==> 2
print(my_math.math_operation(4, 2, "/")) # ==> 2.0
print(my_math.math_operation(4, 2, "*")) # ==> 8
```

Есть класс MathBase и метод math_operation умеющий работать с операциями + и -. Задача класса MathUpgrated расширить функционал метода math_operation операциями * и /.

В новом методе класса MathUpgrated можно продублировать код с базового класса, но копирование кода в программировании — плохая практика (методология DRY). Это значит, что с нового метода нам нужно сохранить возможность обратиться к родительским методам. Для этого в Python есть функция super(). Если ее вызвать внутри метода, мы получим такой же **объект**, как **self**, но с методами родительского класса.

```
# ...            # Получаем self, но с методами родительского класса - MathBase:
parent_class_object = super()

            # Вызываем метод "make_math_operation" на родительском классе, передавая все нужные
аргументы            result = parent_class_object.make_math_operation(number_1, number_2,
operation)            # Возвращаем результат            return result# ...
```

Обратите внимание: super возвращает объект **следующего класса по MRO.**

Закрепление материала

- Что такое наследование?
- Что такое множественное наследование?
- Каким образом в Python указать, что один класс наследуется от другого класса? От нескольких других классов?
- Что такое MRO?
- Каким образом получить доступ к методу базового класса, если он был переопределён в данном классе?

Дополнительное задание

Задание
Создайте иерархию классов транспортных средств. В общем классе опишите общие для всех транспортных средств поля, в наследниках – специфичные для них. Создайте несколько экземпляров. Выведите информацию о каждом транспортном средстве.

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Самостоятельная деятельность учащегося

Задание 1
Создайте класс Editor, который содержит методы view_document и edit_document. Пусть метод edit_document выводит на экран информацию о том, что редактирование документов недоступно для бесплатной версии. Создайте подкласс ProEditor, в котором данный метод будет переопределён. Введите с клавиатуры лицензионный ключ и, если он корректный, создайте экземпляр класса ProEditor, иначе Editor. Вызовите методы просмотра и редактирования документов.

Задание 2
Опишите классы графического объекта, прямоугольника и объекта, который может обрабатывать нажатия мыши. Опишите класс кнопки. Создайте объект кнопки и обычного прямоугольника. Вызовите метод нажатия на кнопку.

Задание 3
Создайте иерархию классов с использованием множественного наследования. Выведите на экран порядок разрешения методов для каждого из классов. Объясните, почему линеаризации данных классов выглядят именно так.

Рекомендуемые ресурсы

Документация Python
https://docs.python.org/3/tutorial/classes.html#inheritance
https://docs.python.org/3/tutorial/classes.html#multiple-inheritance
https://www.python.org/download/releases/2.3/mro/

Статьи в Википедии о ключевых понятиях, рассмотренных на этом уроке
https://ru.wikipedia.org/wiki/Наследование_(программирование)
https://ru.wikipedia.org/wiki/Множественное_наследование
https://ru.wikipedia.org/wiki/C3-линеаризация

Статья о порядке разрешения методов в Python
http://habrahabr.ru/post/62203/

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com