Методические указания

Урок 13.1. Основы ООП

Задачи урока:

- Познакомиться с понятием наследование

0. Подготовка к уроку

До начала урока преподавателю необходимо:

1) Просмотреть, как ученики справились с домашним заданием
2) Прочитать методичку

1. Наследование классов

Учитель: На прошлом занятии мы с вами познакомились с понятие класс и как создавать экземпляры класса. Давайте немного устно повторим материал

1. *Что такое класс?*
2. *Как создается класс*
3. *Что такое объект класса*
4. *Для чего нужен метод init*
5. *Что такое метод класса*
6. *Для чего нужен self*

Учитель: Теперь пришло время узнать, что такое наследование классов.
В Python все классы наследуются от класса object, обладающего некоторыми атрибутами по умолчанию (например, __init__, __doc__, __str__ и т.д.).
Дочерние классы могут изменять поведение атрибутов класса-родителя, переопределив (англ. Override) их. При этом, обычно, дочерний класс дополняет родительский метод, добавив свой код после кода родителя (используя функцию super(), предоставляющую ссылку на родительский класс).

Предположим, нам необходимо создать класс транспорта, но видов транспорта может быть больше количество: легковые машины, грузовые, общественный транспорт и т.п. Эти виды имеют как общие, так и строго индивидуальные характеристики. Как пример грузовой

автомобиль может иметь возможность выгружать перевозимый груз(самосвал), автобус же например, имеет большое количество посадочных мест и принадлежит не частному лицу, а предприятию и многое другое. Как же поступить в данной ситуации. Тут нам поможет возможность классов наследоваться друг от друга. Например мы можем создать общий класс транспорта(родительский), а потом создать дочерние классы. Дочерним классам будут доступны все поля и методы родительского класса.
Разберем пример

```
class Transport:
    def __init__(self, speed, color):
        self.speed = speed
        self.color = color

    def beep(self):
        print('beep')
```

Создали родительский класс и указали, что у транспорта есть скорость, цвет и возможность говорить бип
Теперь необходимо создать дочерние классы. Для указания, что класс является дочерним, необходимо при указании названия класса, в скобках указать класс от которого мы наследуемся.

```
class Car(Transport):
    def __init__(self, speed, color):
        super().__init__(speed, color)
```

В классе Car мы указали, что он наследуется от класса Transport, а также в методе инициализации указали вызов super(), в котором сослались на родительский класс. Теперь укажем для класса пару переменных и какой нибудь метод выведем в консоль.

```
class Transport:
    def __init__(self, speed, color):
        self.speed = speed
        self.color = color

    def beep(self):
        print('beep')

class Car(Transport):
    def __init__(self, speed, color, owner):
        super().__init__(speed, color)
        self.owner = owner
```

```
    def say_owner(self):
        print(f'Владелец {self.owner}')


car1 = Car(100, 'yellow', 'Василий')
print(car1.color)
print(car1.speed)
print(car1.owner)
car1.beep()
car1.say_owner()
```

Из данного примера мы видим, что объект класса Car имеет доступ не только к свойствам и методам своего класса, но и родительского.

Давайте для закрепления еще один дочерний класс

```
class Transport:
    def __init__(self, speed, color):
        self.speed = speed
        self.color = color

    def beep(self):
        print('beep')

class Car(Transport):
    def __init__(self, speed, color, owner):
        super().__init__(speed, color)
        self.owner = owner

    def say_owner(self):
        print(f'Владелец {self.owner}')

class Bus(Transport):
    def __init__(self, speed, color, seeds):
        super().__init__(speed, color)
        self.seeds = seeds

    def say_seeds(self):
        print(f'Кол-во мест {self.seeds}')


car1 = Car(100, 'yellow', 'Василий')
print(car1.color)
print(car1.speed)
print(car1.owner)
car1.beep()
car1.say_owner()
bus1 = Bus(60, 'green', 33)
bus1.say_seeds()
```

Учитель: В Python существует, также понятие множественного наследования.

Python поддерживает концепцию множественного наследования, т.е. позволяет создать класс, имеющий нескольких родителей. В данном случае список наследуемых классов перечисляется при объявлении класса

Множественное наследование - когда класс наследуется более, чем от одного родительского класса. Также родительский класс в котором требуется указать больше данных требуется указать первым.

```
class Transport:
    def __init__(self, speed, color):
        self.speed = speed
        self.color = color

    def beep(self):
        print('beep')

class Car(Transport):
    def __init__(self, speed, color, owner):
        super().__init__(speed, color)
        self.owner = owner

    def say_owner(self):
        print(f'Владелец {self.owner}')


class SportCar(Car, Transport):
    pass


car1 = SportCar(100, 'yellow', 'Иван')
car1.beep()
car1.say_owner()
```

В данном примере мы создали класс,который наследуется от двух предыдущих классов. Несмотря на то что класс SportCar пустой, он наследует все методы и атрибуты от родительских классов.

Если же у нас в классе Car будет метод, с таким же названием как у другого родительского класса, то метод будет переопределен и вызовется метод из класса Car

```
class Transport:
    def __init__(self, speed, color):
        self.speed = speed
        self.color = color

    def beep(self):
        print('beep')

class Car(Transport):
```

```
  def __init__(self, speed, color, owner):
    super().__init__(speed, color)
    self.owner = owner

  def say_owner(self):
    print(f'Владелец {self.owner}')

  def beep(self):
    print('Hello')


class SportCar(Car, Transport):
  pass


car1 = SportCar(100, 'yellow', 'Иван')
car1.beep()
car1.say_owner()
```

2. Решение задач

Задача 1

Написать родительский человека и дочерний класс ученика.

Решение

```
class Person:
  def __init__(self, name, lname, age):
    self.name = name
    self.lname = lname
    self.age = age

  def say_name(self):
    print(f'Привет меня зовут {self.lname} {self.name}')

  def say_age(self):
    print(f'Мой возраст {self.age}')


class Student(Person):
  def __init__(self, name, lname, age, class_num):
    super().__init__(name, lname, age)
    self.class_num =  class_num

  def say_class_num(self):
    print(f'Я учюсь в {self.class_num} классе')



student1 = Student('Иван', 'Петров', 14, 9)
student1.say_class_num()
```

Дополнительно

Если на уроке остается время, то ученикам можно предложить начать прорешивать домашнее задание.

Домашняя работа

Задача 1

Реализовать родительский класс человека, а также дочерние классы директора, преподавателя и ученика. Описать для каждого класса необходимые свойства и методы. Важно: директор помимо своих обязанностей может также и преподавать(множественное наследование)

Решение

```
class Person:
  def __init__(self, name, lname, age):
    self.name = name
    self.lname = lname
    self.age = age

  def say_name(self):
    print(f'Привет меня зовут {self.lname} {self.name}')

  def say_age(self):
    print(f'Мне {self.age} лет')

class Teacher(Person):
  def __init__(self, name, lname, age, lesson):
    super().__init__(name, lname, age)
    self.lesson = lesson

  def teaching(self):
    print(f'Я преподаю предмет {self.lesson}')

class Student(Person):
  def __init__(self, name, lname, age, class_number):
    super().__init__(name, lname, age)
    self.cls_number = class_number

  def say_cls_number(self):
    print(f'Я учусь в {self.cls_number} классе')

class Director(Teacher, Person):
  def say(self):
    print('Я директор школы')
```