Тема 11. Основы объектно-ориентированного программирования (ООП)

Объектно-ориентированный подход в программировании подразумевает наличие в программе классов, объектов, взаимодействие объектов между собой, в результате которого меняются их свойства.

Класс – это абстрактный тип данных. С помощью класса описывается некоторая сущность (ее характеристики и возможные действия). Например, класс может описывать студента, автомобиль и т. д. Описав класс, мы можем создать его воплощение – объект. **Объект** – это конкретный экземпляр (представитель) класса.

Объект в программе можно создать лишь на основе какого-нибудь класса. Классы могут располагаться либо в начале кода программы, либо импортироваться из других файлов-модулей (также в начале кода).

11.1. Создание класса

Для создания классов предусмотрена инструкция **class**. Она состоит из строки заголовка и тела. Заголовок состоит из ключевого слова **class**, имени класса. После имени класса в скобках можно указать названия суперклассов, на основе которых создается класс. Мы будем создавать свои классы без суперклассов, поэтому скобки в конце ставить не будем.

Тело класса состоит из блока инструкций. Тело должно быть обязательно написано с отступами согласно синтаксису языка **Python**.

Атрибуты класса – это имена переменных вне функций (**полей**) и имена функций. Эти атрибуты наследуются всеми объектами, созданными на основе данного класса. Атрибуты характеризуют свойства и поведение объекта. Объекты могут иметь атрибуты, которые создаются в теле метода, если этот метод будет вызван для конкретного объекта.

Атрибуты у экземпляра класса указываются через точку.

Общий вид класса можно представить следующим образом:

```
class <имя класса>[(<суперклассы>)]:

    <переменная1>=<значение1>
    <переменная2>=<значение2>
...
    <переменнаяn>=<значение n>

    def <имя метода>([self,<параметры>])
        self.<переменная>=<значение>
        ...
    ...
```

Пример:

Допустим, нам нужна конструкция для хранения информации об объекте «студент». Важными свойствами являются ФИО, пол, возраст, факультет, курс, специальность, группа. Тогда мы можем создать следующий класс:

```
class student:

    full_name = 'Не определено'
    sex = 'Не определено'
    age = 0
    faculty = 'Не определено'
    year = 0
    specialty = 'Не определено'
    group = 'Не определено'
```

11.2. Создание объекта

Объекты (экземпляры класса) создаются следующим образом:

<имя переменной>=<имя класса()>

В записи скобки ставятся обязательно. После выполнения такой команды в программе появляется объект, доступ к которому можно получить по имени переменной, связанной с ним. При создании объект получает атрибуты его класса, т. е. объекты обладают характеристиками, определенными в их классах.

Количество объектов, которые можно создать на основе того или иного класса, не ограничено.

Объекты одного класса имеют схожий набор атрибутов, а вот значения этих атрибутов у каждого объекта могут быть разными. Другими словами, объекты одного класса похожи, но индивидуально различны.

Пример:

Создадим объект класса ***student***

```
st = student()
```

Все свойства объекта ***st*** примут значения по умолчанию, которые указаны в описании класса. Таким образом, мы создали класс и на его основе создали один объект. Количество объектов может быть любое, причем объекты можно добавлять в списки, кортежи и т. д.

11.3. Изменение и получение атрибутов

После создания объекта к его свойствам обращаются через точку. Работаем с тем же объектом ***st*** – экземпляром класса ***student***.

Допустим, нам нужно вывести на экран для объекта ***st*** свойства ФИО и возраст. Поступаем следующим образом:

```
print('ФИО =', st.full_name)
print('Возраст =', st.age)
```

```
ФИО = Не определено
Возраст = 0
```

Обратите внимание, что поля получили значения по умолчанию, указанные при описании класса.

Теперь нам нужно поменять свойства. Процесс аналогичный, только теперь уже присваиваем новые значения.

Пример:

Нам нужно изменить ФИО объекта

```
st.full_name = 'Иванов Иван Иванович'
```

Обращение к полю:

```
print(st.full_name)
```

В этом примере изменения значения поля мы выполнили непосредственно в коде.

А теперь зададим новое ФИО с клавиатуры после запуска программы

```
st.full_name = input('Введите новые ФИО для объекта: ')
```

11.4. Методы

Методы в классах, по сути, являются теми же функциями. Отличие заключается в том, что методы принимают один обязательный параметр – **self**. Он нужен для связи с объектом.

Методы могут изменять существующие свойства объекта, создавать новые свойства объекта (на практике применять не рекомендуется), выполнять другие действия над объектами. Методу необходимо «знать», данные какого объекта ему предстоит обрабатывать. Для этого ему в качестве первого (а иногда и единственного) аргумента передается имя переменной, связанной с объектом (можно сказать, передается ссылка на объект). Чтобы в описании класса указать передаваемый в дальнейшем объект, используется параметр **self**.

Вызов метода для конкретного объекта в основном блоке программы выглядит следующим образом:

<объект>.<имя_метода(...)>

Здесь под словом *объект* подразумевается переменная, связанная с ним. Это выражение преобразуется в классе, к которому относится объект, в выражение:

<имя_метода(объект, ...)>

т. е. имя конкретного объекта подставляется вместо параметра **self**.

Пример:

Далее работать будем с тем же классом ***student***.

Так как студент с каждым годом меняет курс, может поменять специальность, а также группу, то напишем для этой цели соответствующие методы. Кроме того, создадим отдельный метод для вывода свойств объекта на экран.

Методы задаются в описании класса.

```
    # метод для повышения курса
    def upyear(self):
        self.year += 1

    # метод для вывода всех свойств объекта на экран
    def propprint(self):
        print('ФИО =', self.full_name)
        print('Пол =', self.sex)
        print('Возраст =', self.age)
        print('Факультет =', self.faculty)
        print('Курс =', self.year)
        print('Специальность =', self.specialty)
        print('Группа =', self.group)
```

Эти методы не требуют указания дополнительных параметров. А методы для смены специальности и смены группы требуют как минимум по одному параметру – это указание новой специальности и новой группы. Параметры мы должны указать в скобках, через запятую после **self**.

```
    # метод для смены группы
    def changegroup(self, newgroup):
        self.group = newgroup

    # метод для смены специальности
    def changespec(self, newspec):
        self.specialty = newspec
```

Выведем значения всех полей на экран описанным выше методом

```
st.propprint()
```

В результате увидим следующее:

```
ФИО = Не определено
Пол = Не определено
Возраст = 0
Факультет = Не определено
Курс = 0
Специальность = Не определено
Группа = Не определено
```

Для изменения группы вызовем соответствующий метод

```
st.changegroup('15 ИиИКТ')
```

В скобках указано название новой группы.

Для изменения специальности также вызовем соответствующий метод

```
st.changespec('Информатика')
```

В скобках указано название новой специальности.

Выведем снова свойства объекта

```
st.propprint()
```

В результате получим следующее:

```
ФИО = Не определено
Пол = Не определено
Возраст = 0
Факультет = Не определено
Курс = 0
Специальность = Информатика
Группа = 15 ИиИКТ
```

11.5. Конструкторы

Выше мы уже говорили о том, что при создании экземпляра класса (объекта) значения полей задаются по умолчанию, а после мы можем менять их значения. Но это не всегда удобно. Часто корректнее было бы указывать значения некоторых полей сразу, при создании экземпляра класса. Для этой цели служит конструктор класса.

Конструктор – это метод, который вызывается автоматически в момент создания объекта. В нем необходимо создать свойства класса, если они не были определены.

Конструктор в Python всегда имеет имя **__init__** . Обратите внимание, в начале и конце стоит по **2 знака нижнего подчеркивания**. Не забываем, что первый параметр в методе – это **self**, и конструктор не является исключением. Конструктор нужно писать в самом начале класса.

Общий вид

def *__init__(**self**, <параметр1>, <параметр2>, ...,<параметрN>):*
<атрибут1>=<параметр1>
<атрибут2>=<параметр2>
……
<атрибутN>=<параметрN>

Будем использовать тот же класс ***student***. Создадим для него конструктор.

```
    def __init__(self, Full_name, Sex, Age, Faculty, Year, Specialty, Group):
        self.full_name = Full_name
        self.sex = Sex
        self.age = Age
        self.faculty = Faculty
        self.year = Year
        self.specialty = Specialty
        self.group = Group
```

Обратите внимание, имена параметров такие же, как имена атрибутов, только написанны с большой буквы. Так делают для удобства. Но на самом деле имена параметров могут быть любыми, но не совпадающими с именами атрибутов.

Удалим из класса ***student*** значения полей по умолчанию.

Полный класс будет выглядеть следующим образом:

```
class student:
    # конструктор
    def __init__(self, Full_name, Sex, Age, Faculty, Year, Specialty, Group):
        self.full_name = Full_name
        self.sex = Sex
        self.age = Age
        self.faculty = Faculty
        self.year = Year
        self.specialty = Specialty
        self.group = Group

    # метод для повышения курса
    def upyear(self):
        self.year += 1

    # метод для вывода всех свойств объекта на экран
    def propprint(self):
        print('ФИО =', self.full_name)
        print('Пол =', self.sex)
        print('Возраст =', self.age)
        print('Факультет =', self.faculty)
        print('Курс =', self.year)
        print('Специальность =', self.specialty)
        print('Группа =', self.group)

    # метод для смены группы
    def changegroup(self, newgroup):
        self.group = newgroup

    # метод для смены специальности
    def changespec(self, newspec):
        self.specialty = newspec
```

Теперь для создания экземпляра класса используем ту же запись, что и прежде, но в скобках мы обязательно должны указать значения полей.

```
st = student('Иванов И.И.', 'мужской', 18, 'ФМ', 1, 'Информатика', '15ИиИКТ')
st.propprint()
```

Результат:

```
ФИО = Иванов И.И.
Пол = мужской
Возраст = 18
Факультет = ФМ
Курс = 1
Специальность = Информатика
Группа = 15ИиИКТ
```

Попытка создания объекта без указания начальных значений полей

```
st = student()
```

приведет к ошибке.

Все потому, что теперь у нас не указаны значения по умолчанию. Их нужно указать в конструкторе следующим образом:

```
    # конструктор
    def __init__(self, Full_name='Не определено', Sex='Не определено', Age=0,\
                 Faculty='Не определено', Year=0, Specialty='Не определено', \
                 Group='Не определено'):
        self.full_name = Full_name
        self.sex = Sex
        self.age = Age
        self.faculty = Faculty
        self.year = Year
        self.specialty = Specialty
        self.group = Group
```