**Тема «классы»**

В Python все является объектом. Это очень расплывчатое определение, означающее, что каждый объект в Python имеет метод и значение по той причине, что все объекты базируются на классах. Даже переменная x=4 имеет свои методы. Класс – это проект объекта.

Подход, при котором для разработки используются классы, называется объектно-ориентированным.

**Создание класса**

Синтаксис создания класса следующий:

|  |  |
| --- | --- |
| **class** **Vehicle**:  """docstring"""    **def** **\_\_init\_\_**(self):  """Constructor"""  **pass** | **class** **Vehicle(**object**)**:  """docstring"""    **def** **\_\_init\_\_**(self):  """Constructor"""  **pass** |

Для того, чтобы создать класс, необходимо написать class и наименование класса. В примере код отличается элементом (object). Разница заключается в том, что классы могут друг от друга наследоваться. Про наследование будет описано позднее, но на данном моменте следует запомнить, что есть два варианта задания класса.

Метод \_\_init\_\_ вызывается всякий раз, когда создается новый объект (экземпляр класса) класса. Этот метод вызывает единожды и не может быть вызван вторично.

*Замечание*: функции становится методом, как только она находится внутри класса.

*Замечание*: каждый метод должен иметь как минимум один аргумент (self).

Немного расширим исходный класс

|  |
| --- |
| **class** **Vehicle**:  """docstring"""    **def** **\_\_init\_\_**(self, color, doors, tires):  """Constructor"""  self.color = color  self.doors = doors  self.tires = tires    **def** **brake**(self):  """  Stop the car  """  **return** "Braking"    **def** **drive**(self):  """  Drive the car  """  **return** "I'm driving!" |

Теперь у элемента появилось три атрибута (self.color, self.doors и self.tires) и два метода (brake() и drive()). Атрибуты объекта описывают объект, а методы описывают его поведение.

**Self**

В примерах выше часто использовалось слово self. Оно является необходимым со следующей точки зрения

Классам необходим способ, чтобы ссылаться на самих на себя. Это позволяет элементам быть независимым друг от друга. Буквально слово self означает способ описание любого объекта. Необходимость self легко показать на примере.

|  |
| --- |
| car = Vehicle("blue", **5**, **4**)  print(car.color)    truck = Vehicle("red", **3**, **6**)  print(truck.color) |

Наследование

Обратимся к самому первому примеру, когда рассматривались две записи создания класса. Для каждого класса можно создать подкласс, которые основан на родительском классе и как-то дополняет его (добавляются новые атрибуты и функции). Способность одного класса получать или наследовать свойства другого класса называется наследованием.

Попробуем создать подкласс Car, основанный на классе Vehicle

|  |
| --- |
| **class** **Car**(Vehicle):  """  The Car class  """  **def** **repair**(self):  **return** "The car is under repair"  **def** **brake**(self):  """  Override brake method  """  **return** "The car class is breaking slowly!"      **if** \_\_name\_\_ == "\_\_main\_\_":  car = Car("yellow", **2**, **4**, "car")  car.brake()  'The car class is breaking slowly!'  car.drive()  "I'm driving"  car.repair()  "The car is under repair " |

Заметьте, что методы \_\_init\_\_ и drive не использовались. Это связано с тем, что они взяты из класса Vehicle. Переписывать данные методы стоило только тогда, когда необходимо их переопределить. В то же время произошло переопределение метода brake.

**ООП**

ООП подразумевает использование классов и стоит на трех столпах: наследование, инкапсуляция и полиморфизм. Но сегодняшним занятии произошло знакомство с наследованием.