Объектно-ориентированное программирование

Николай Вяххи

Санкт-Петербургский Государственный Университет

2 апреля 2010 г.

ООП

Объектно-ориентированное программирование (ООП) – парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

Класс – это тип, описывающий устройство объектов.

Объект – сущность класса.

Пример класса с методами

```
1  class Car(object):
2      def brake(self):
3          print ' Brakes '
4
5      def accelerate(self):
6          print ' Accelerating '
7
8  my_car = Car()
9  my_car.accelerate()
10 my_car.accelerate()
11 my_car.brake()
```

Пример встроенных классов

Все базовые элементы (строки, словари, числа, функции) в Python – тоже объекты.

```
1  x = math.sqrt(2.0)
2  print x.as_integer_ratio()
3  # (6369051672525773L, 4503599627370496L)
4
5  s = ' Hello world '
6  print s.split( ' o ' )
7  # ['Hell', ' w', 'rld']
```

Свойства

```
1  class Car(object):
2      speed = 0
3
4      def brake(self):
5          self.speed = 0
6
7      def accelerate(self):
8          self.speed += 10
9
10 c = Car() # c.speed == 0
11 c.accelerate() # c.speed == 10
12 c.accelerate() # c.speed == 20
13 c.brake() # c.speed == 0
```

Private vs Public

Свойства, скрытые от внешнего мира (private) именуются с двумя подчеркиваниями.

```
1  class Car(object):
2      speed = 0
3      __speed = 0
4
5  c = Car()
6  c.speed = 10 # OK
7  c.__speed = 10 # Error!
```

Свойства

Все свойства должны быть private!

Вы не можете изменить скорость машины своим желанием, но можете нажать на газ или на тормоз.

Getters and Setters

Если нужно иметь доступ к свойству, определите для него getter (accessor) и/или setter (mutator).

```
1  class Car(object):
2      __speed = 0
3      def get_speed(self):
4          return self.__speed
5      def set_speed(self, speed):
6          assert isinstance(speed, int)
7          self.__speed = min(speed, 200)
8
9  c = Car()
10 c.set_speed(60)
11 print c.get_speed()
```

Getters and Setters

Удобно тем, что логика локализуется в одном месте. Иначе, пришлось бы так:

```
1  class Car(object):
2      speed = 0
3
4  c = Car()
5  c.speed = min(speed1, 200)
6  # ...
7  c.speed = min(speed2, 200)
8  # ...
9  c.speed = min(speed3, 200)
```

Конструктор

Если при создании объекта хочется производить инициализирующие действия, поможет конструктор:

```
1  class Car(object):
2      def __init__(self, car_type = 'truck'):
3          self.__type = car_type
4          self.__speed = 0
5          while True:
6              filename = random_string() + '.log'
7              if not os.path.isfile(filename):
8                  self.__logfile = open(filename, 'w')
9                  break
10 c1 = Car()
11 c2 = Car('limousine')
```

Свойства класса и свойства объекта

```
1  class Car(object):
2      str = ' class string '
3      def __init__(self):
4          self.str = ' object string '
5
6  c = Car()
7  Car.str = ' new class string '
8  print c.str # object string
9  print Car.str # new class string
10 print c.__class__.str # new class string
11
12 print c.__dict__ # {'str': 'object string'}
13 print Car.__dict__ # ...
```

Сборка мусора

Удалять объекты не надо, они удалятся сами когда станут не нужны программе (пропадут все ссылки на них).

Если хочется что-то закрыть перед удалением, поможет деструктор:

```
1  class Car(object):
2
3      ...
4
5      def __del__(self):
6          self.__logfile.close()
```

Строковое представление

Чтобы объекты могли себя печатать, достаточно определить метод __str__:

```
1  class Car(object):
2      ...
3
4      def __str__(self):
5          return " Car with speed  " + str(self.__speed)
6
7  c = Car()
8  print c
```

Наследование

Один класс может расширять другой класс.

```
1  class Limo(Car):
2      def __init__(self, color = 'black'):
3          self.__color = color
4          Car.__init__('limousine')
5
6  class Truck(Car):
7      def __init__(self):
8          Car.__init__('truck')
9
10 l = Limo()
11 t = Truck()
```

Наследование

Наследование отвечает на вопрос „является“.

Например: „Лимузин **является** машиной“.

Если у вас „Машина **содержит** двигатель“, то тогда класс машина должна **содержать свойство** двигатель, а не наследоваться от него.

Полиморфизм

Полиморфизм - взаимозаменяемость объектов с одинаковым интерфейсом.

```
1  a = (1, 2, 3)
2  b = ['a', 'b', 'c']
3  c = {1 : 'hello', 2 : 'world'}
4  ls = [a, b, c]
5  for x in ls:
6      print x[1],
7
8  # 2 b hello
```

Полиморфизм

```
1  class Car(object):
2      def accelerate(self):
3          self.set_speed(self.get_speed() + 10)
4
5  class SportsCar(Car):
6      def accelerate(self):
7          self.set_speed(self.get_speed() + 20)
8
9  cars = [Car(), SportsCar(), Car()]
10 for car in cars:
11     car.accelerate()
```

Duck Typing (Утиная типизация)

> Если что-то крякает как утка, плавает как утка и выглядит как утка, то это, вероятно, утка и есть.

В Python нет интерфейсов как в Java, поэтому полиморфизму подвержены все объекты с одинаковыми методами, независимо от наследования.

Принципы ООП

- Абстракция данных
- Инкапсуляция
- Наследование
- Полиморфизм

Полезно знать

Классы – тоже объекты!

Их предок – класс „тип“.

Полезно знать

```
1  class Car(object): pass
2  c = Car()
3
4  print c # <__main__.Car object at 0x0000000002086DA0>
5
6  print Car # <class '__main__.Car'>
7  print type(c) #  <class '__main__.Car'>
8  print c.__class__ # <class '__main__.Car'>
9
10 print type(Car) # <type 'type'>
11 print type(type(c)) # <type 'type'>
12 print Car.__class__ # <type 'type'>
13
14 print Car.__bases__ # (<type 'object'>,)
15
16 print type(type(Car)) # <type 'type'>
17 print type(Car).__bases__ # (<type 'object'>,)
```

Полезные ссылки

Python Programming/Object-oriented programming
http://en.wikibooks.org/wiki/Python_Programming/Object-oriented_programming

Python Types and Objects
http://www.cafepy.com/article/python_types_and_objects

Python Attributes and Methods
http://www.cafepy.com/article/python_attributes_and_methods/