2 - Object-Oriented Programming in Python

November 30, 2015

1 Syntax

- In Python, **everything** is an object (data types, function, classes . . .).
- **Inheritance** and **multiple inheritance** exist in Python.
- We can make our Python files act as **standalone executable** or **reusable modules**. This will set the Python file as the **‘main’** program.

  Example:

```
In [ ]: if __name__ == "__main__": # The ’main’ function
            #do_your_things_here
            print "WHATEVER"
```

- The **__init__** method is used to define a **class constructor**.

  Example:

```
In [ ]: class Animal(object):
            def __init__(self, *args):
                # do something with args
                pass
```

- The keyword **self** always refer to the instance of a class (not the class itself). It is always passed as an argument in class functions.
  - **self.foo** refers to a class attribute.
  - **self.foo()** refers to a class method.
  - **def foo(self):** is a standard class method definition

  Example:

```
In [ ]: class Animal(object):
            def __init__(self, name, age, occupation):
                self.name = name
                self.age = age
                self.occupation = occupation
                self.introduce()

            def introduce(self):  # this is called by self.introduce()
                print self.name, self.age, self.occupation

        myAnimal = Animal("Patrick", 22, "Architect")
        # myAnimal.name
        # myAnimal.age
        # myAnimal.occupation
        # myAnimal.introduce()
```

- Private attributes or functions must be prefixed by one or multiple underscore. Private functions don't have to be documented (at least for others).

Example:

```
In [ ]: class Animal(object):
            def __init__(self, name, age, occupation):
                # All arguments private now
                self.__name = name
                self.__age = age
                self.__occupation = occupation
                self.__introduce()

            def __introduce(self): # introduce is private
                print self.__name, self.__age, self.__occupation

        # can't access introduce from outside anymore !
        myAnimal = Animal("Patrick", 22, "Architect")
        # myAnimal.__name
        # myAnimal.__age
        # myAnimal.__occupation
        # myAnimal.__introduce()
```

2 Functions

```
In [ ]: # Definition
        def foo(bar):
            print bar
```

```
In [ ]: # Function call
        foo("hello!")
```

Functions can return anything / multiple things

```
In [ ]: def foo():
            return "John", "Oliver"  # returns tuple

        firstName, lastName = foo()  # uses tuple unwrapping
        print firstName, lastName
```

```
In [ ]: class Dog(object):
            def __init__(self):
                pass

        def foo():
            obj1 = Dog()
            obj2 = Dog()
            return obj1, obj2

        dog1, dog2 = foo()
        print dog1, dog2
```

Functions can accept anything as argument (type overloading)

```
In [ ]: # If you don’t do type-specific things, type is not enforced in Python.
        # Every object with an __str__ method defined can be printed in Python, so the following calls
        foo(5)     # int
        foo(5.0)   # float
        foo([1, 2, 3, 4]) #list
        foo({’John’: [22, ’architect’], #dict
             ’Tom’ : [25, ’senior investor’]
            })
```

```
In [ ]: # don’t write this ...
        def foo_int(bar):
            print "INT: %d" % bar
        def foo_string(bar):
            print "STR: %s" % bar

        bar = "ok"
        if isinstance(bar, int):
            foo_int(bar)
        elif isinstance(bar, str):
            foo_string(bar)
```

```
In [ ]: # ... write this
        def foo(bar):
            if type(bar) == int:
                print "INT: %d" % bar
            elif type(bar) == str:
                print "STR: %s" % bar

        bar = 5
        # bar = "hello"
        foo(bar)
```

Optional arguments

```
In [ ]: # Single optional arguments
        def foo(bar, opt=’’):
            if opt:
                print "Optional !"
            print bar

        bar = 5
        opt = ’anything’
        foo(bar)
        # foo(bar, opt)
```

```
In [ ]: # Multiple optional arguments
        def foo(bar, **kwargs):
            for index, arg in enumerate(kwargs):
                print "Arg %d: %s" % (index, arg)
        bar = 5
        foo(bar, opt1=’arg1’, opt2=’arg2’, opt3=’arg3’)
```

3 Classes

3.1 Basics

```
In [ ]: # class definition
        class Dog(object):          # Every class inherits from 'object'.
            def __init__(self):     # Every class has an __init__ method (the 'constructor').
                pass                # Every method of a class takes the 'self' argument.
```

```
In [ ]: # create an instance of the class
        myDog = Dog()
```

Adding a new function

```
In [ ]: # class definition
        class Dog(object):
            """ Class defining a dog. """ # Docstring: code documentation
            def __init__(self):
                self.bark()                 # Call bark at initialization (self.bark() means the bark met

            def bark(self):                 # New method: bark
                print "BARK!"


        # main function
        if __name__ == '__main__':
            myDog = Dog()                   # create an instance of the class
```

3.2 Inheritance

3.2.1 Normal inheritance

```
In [ ]: # class definitions
        class Animal(object):
            """ Class defining an animal. """
            def __init__(self, name):
                self.name = name
                self.emit_sound()

            def emit_sound(self):
                if self.animal == "dog":
                    self.bark()
                elif self.animal == "cat":
                    self.meow()

            def introduce(self):
                print "I am %s the %s" % (self.name, self.animal)


        class Dog(Animal):
            """ Class defining a dog. """
            def __init__(self, name):
                self.animal = "dog"
                super(Dog, self).__init__(name)  #super --> parent object (Animal)

            def bark(self):
                print "BARK!"


        class Cat(Animal):
            """ Class defining a cat. """
            def __init__(self, name):
                self.animal = "cat"
                super(Cat, self).__init__(name)  #super --> parent object (Animal)

            def meow(self):
                print "MEOW!"

        # main function
        if __name__ == '__main__':
            myDog = Dog("Harry") # create an instance of class 'Dog'
            myDog.introduce()

            myCat = Cat("Felix") # create an instance of class 'Cat'
            myCat.introduce()
```

3.2.2 Ancestor inheritance

```
In [ ]: # class definitions
        class First(object):
            def __init__(self):
                print "first"

        class Second(First):     # Second inherits from First
            pass

        class Third(Second):     # Third inherits from Second
            pass

        # main function
        if __name__ == '__main__':
            myThird = Third()
```

```
In [ ]: # class definitions
        class First(object):
            def __init__(self):
                print "first"

        class Second(First):     # Second inherits from First
            def __init__(self):
                super(Second, self).__init__()
                print "second"

        class Third(Second):     # Third inherits from Second
            def __init__(self):
                super(Third, self).__init__()
                print "third"

        # main function
        if __name__ == '__main__':
            myThird = Third()
```

3.2.3 Multiple inheritance

```
In [ ]: # an easy one ..
        class First(object):
            def __init__(self):
                print "first"


        class Second(First):
            def __init__(self):
                print "second"


        class Third(First):
            def __init__(self):
                print "third"


        class Fourth(Second, Third):
            def __init__(self):
                super(Fourth, self).__init__()
                print "that’s it"


        # main function
        if __name__ == ’__main__’:
            myFourth = Fourth()
```

```
In [ ]: # a more complex one ..
        class First(object):
            def __init__(self):
                print "first"

            def save(self):
                print "Saving First"

        class Second(First):
            def __init__(self):
                print "second"

            def save(self):
                print "Saving Second"

        class Third(First):
            def __init__(self):
                print "third"

            def save(self):
                print "Saving Third"

        class Fourth(Second, Third):
            def __init__(self):
                super(Fourth, self).__init__() # calls Second’s init method
                print "that’s it"
        #     def save(self):
```

```
        #         Third().save()
        #     def save(self):
        #         super(Third, self).save()

        # main function
        if __name__ == '__main__':
            myFourth = Fourth()
            myFourth.save()
```

- Method resolution order: http://python-history.blogspot.com/2010/06/method-resolution-order.html

3.2.4 A complete example of inheritance

Base: Dog and SuperDog

```
In [ ]: # class definitions
        class Dog(object):
            """ A normal dog. """
            animal = "dog"
            def __init__(self, name):
                self.name = name
                self.moves = []

            def moves_setup(self):
                self.moves.append('walk')
                self.moves.append('run')

            def get_moves(self):
                return self.moves

            def introduce(self):
                print "I am %s the %s" % (self.name, self.animal)
                print "My moves are:",
                print self.moves


        class SuperDog(Dog):
            """ This dog can fly."""
            animal = "superdog"
            def __init__(self, name):
                super(SuperDog, self).__init__(name)

            def moves_setup(self):
                super(SuperDog, self).moves_setup() # set moves from parent class
                self.moves.append('fly')            # new move: 'fly' !


        # main function
        if __name__ == '__main__':
            dog = Dog("Freddy")
            dog.moves_setup()
            dog.introduce()

            print
            superDog = SuperDog("John")
            superDog.moves_setup()
            superDog.introduce()
```

Adding Animal class

```
In [ ]: # class definitions
        class Animal(object):           # new class: Animal
            """ Any animal."""
            def __init__(self, name):
                self.name = name
                self.moves = []
                self.moves_setup()      # moves_setup() function call moved here
                self.introduce()        # introduce() function call moved here

            def introduce(self):        # introduce() function moved here
                print "I am %s the %s" % (self.name, self.animal)
                print "My moves are:",
                print self.moves


        class Dog(Animal):              # Dog inherits from Animal now
            """ A normal dog. """
            animal = "dog"
            ### __init__ function inherited from parent

            def moves_setup(self):
                self.moves.append(’walk’)
                self.moves.append(’run’)


        class SuperDog(Dog):
            """ This dog can fly."""
            animal = "superdog"
            ### __init__ function inherited from parent

            def moves_setup(self):
                super(SuperDog, self).moves_setup()
                self.moves.append(’fly’)

        # main function
        if __name__ == ’__main__’:
            dog = Dog("Freddy")
            print
            superDog = SuperDog("John")
```

Adding SuperHero class

```
In [ ]: # class definitions
        class SuperHero(object):        # new class: SuperHero
            """ A super hero has some new skills. """
            def taunt(self):
                print "I am more powerful than a normal",
                print self.__class__.__base__.animal
```

```
class Animal(object):
    """ An animal. """
    def __init__(self, name):
        self.name = name
        self.moves = []
        self.moves_setup()      # moves_setup() function call moved here
        self.introduce()        # introduce() function call moved here

    def introduce(self):        # introduce() function moved here
        print "I am %s the %s" % (self.name, self.animal)
        print "My moves are:",
        print self.moves


class Dog(Animal):
    """ A normal dog. """
    animal = "dog"

    def moves_setup(self):
        self.moves.append('walk')
        self.moves.append('run')


class SuperDog(Dog, SuperHero): # SuperDog inherits from Dog AND SuperHero now
    """ This dog can fly."""
    animal = "superdog"

    def moves_setup(self):
        super(SuperDog, self).moves_setup() # Calling 'moves_setup' from Dog
        self.moves.append('fly')

    def taunt(self):
        super(SuperDog, self).taunt()       # Calling 'taunt' from SuperHero

# main function
if __name__ == '__main__':
    dog = Dog("Freddy")
#     dog.taunt()
    print
    superDog = SuperDog("John")
    superDog.taunt()
```

Adding Cat and SuperCat classes

```
In [ ]: # class definitions
        class SuperHero(object):        # new class: SuperHero
            """ A super hero has some new skills. """
            def taunt(self):
                print "I am more powerful than a normal",
                print self.__class__.__base__.animal

        class Animal(object):
            """ An animal. """
            def __init__(self, name):
                self.name = name
                self.moves = []
                self.moves_setup()      # moves_setup() function call moved here
                self.introduce()        # introduce() function call moved here

            def introduce(self):        # introduce() function moved here
                print "I am %s the %s" % (self.name, self.animal)
                print "My moves are:",
                print self.moves


        class Dog(Animal):
            """ A normal dog. """
            animal = "dog"

            def moves_setup(self):
                self.moves.append('walk')
                self.moves.append('run')

        class SuperDog(Dog, SuperHero): # SuperDog inherits from Dog AND SuperHero now
            """ This dog can fly."""
            animal = "superdog"

            def moves_setup(self):
                super(SuperDog, self).moves_setup() # Calling 'moves_setup' from Dog
                self.moves.append('fly')

            def taunt(self):
                super(SuperDog, self).taunt()       # Calling 'taunt' from SuperHero


        class Cat(Animal):
            """ A normal cat. """
            animal = "cat"

            def moves_setup(self):
                self.moves.append('walk')
                self.moves.append('run')
                self.moves.append('climb')

        class SuperCat(Cat, SuperHero):
            """ This cat can fly. """
            animal = "supercat"

            def moves_setup(self):
                super(SuperCat, self).moves_setup()
                self.moves.append('fly')

            def taunt(self):
                super(SuperCat, self).taunt()


        # main function
        if __name__ == '__main__':
            dog = Dog("Patrick")
```

```
            print
            superDog = SuperDog("John")
            superDog.taunt()
            print
            cat = Cat("Ursula")
            print
            superCat = SuperCat("Felix")
            superCat.taunt()
```

Improving overall class design

```
In [ ]: # class definitions
        class SuperHero(object):
            """ A super hero has some new skills. """
            moves = ['fly', 'power punch']

            def taunt(self):
                print "I am more powerful than a normal",
                print self.__class__.__base__.animal

            def power_punch(self):
                print "Hitting bad guys with power punch !"

        class Animal(object):
            """ An animal. """
            def __init__(self, name):
                self.name = name
                self.moves = []
                self.moves_setup()
                self.introduce()

            def moves_setup(self):
                self.moves.append('walk')
                self.moves.append('run')

            def introduce(self):        # introduce() function moved here
                print "I am %s the %s" % (self.name, self.animal)
                print "My moves are:",
                print self.moves


        class Dog(Animal):
            """ A normal dog. """
            animal = "dog"

        class SuperDog(Dog, SuperHero): # SuperDog inherits from Dog AND SuperHero now
            """ This dog can fly."""
            animal = "superdog"

            def __moves_setup(self):
                super(SuperDog, self).moves_setup()  # Calling 'moves_setup' from Dog
                self.moves.extend(SuperHero.moves)   # Adding SuperHero moves

            # removing def taunt(self) here
```

```
class Cat(Animal):
    """ A normal cat. """
    animal = "cat"

class SuperCat(Cat, SuperHero):
    """ This cat can fly. """
    animal = "supercat"

    def __moves_setup(self):
        super(SuperCat, self).moves_setup() # Calling ’moves_setup’ from Cat
        self.moves.extend(SuperHero.moves)  # Adding SuperHero moves

    # removing def taunt(self) here

# main function
if __name__ == ’__main__’:
    dog = Dog("Patrick")
    print
    superDog = SuperDog("John")
    superDog.taunt()
    superDog.power_punch()       # Call to ’power_punch’ from SuperHero
    print
    cat = Cat("Ursula")
    print
    superCat = SuperCat("Felix")
    superCat.taunt()
    superCat.power_punch()       # Call to ’power_punch’ from SuperHero
```