Object Oriented Programming

Classes and objects are the two main aspects of object oriented programming. A **class** creates a new *type* where **objects** are **instances** of the class. An analogy is that you can have variables of type `int` which translates to saying that variables that store integers are variables which are instances (objects) of the `int` class.

`self`

attribute reference.

The `self` in Python is equivalent to the `this` pointer in C++ and the `this` reference in Java and C#.

simplest class

oop_simplestclass.py

```
class Person:
    pass  # An empty block

p = Person()
print(p)
```

Output:

```
$ python oop_simplestclass.py
<__main__.Person instance at 0x10171f518>
```

Methods

oop_method.py

```
class Person:
    def say_hi(self):
        print('Hello, how are you?')

p = Person()
p.say_hi()
# The previous 2 lines can also be written as
# Person().say_hi()
```

Output:

```
$ python oop_method.py
Hello, how are you?
```

`__init__` method

oop_init.py

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

    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('Swaroop')
p.say_hi()
# The previous 2 lines can also be written as
# Person('Swaroop').say_hi()
```

Output:

```
$ python oop_init.py
Hello, my name is Swaroop
```

`self`

The `self` in Python is equivalent to the `this` pointer in C++ and the `this` reference in Java and C#.

Class Variables

Class Variables are shared.
..They can be accessed by all instances of that class.
..A class variable has only one copy.
..A change to a class variable will be seen by all instance.

Object Variables

Object Variables are owned by an instance of the class.
..They are not shared to other instances of the same class.
..They only have the same variable object types and names.

All class members (including the data members) are *public* and all the methods are *virtual* in Python.

Inheritance

oop_subclass.py

```python
class SchoolMember:
    '''Represents any school member.'''
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print('(Initialized SchoolMember: {})'.format(self.name))

    def tell(self):
        '''Tell my details.'''
        print('Name:"{}" Age:"{}"'.format(self.name, self.age), end=" ")
```

```python
class Teacher(SchoolMember):
    '''Represents a teacher.'''
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print('(Initialized Teacher: {})'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Salary: "{:d}"'.format(self.salary))
```

```python
class Student(SchoolMember):
    '''Represents a student.'''
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks = marks
        print('(Initialized Student: {})'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Marks: "{:d}"'.format(self.marks))
```

```python
t = Teacher('Mrs. Shrividya', 40, 30000)
s = Student('Swaroop', 25, 75)

# prints a blank line
print()

members = [t, s]
for member in members:
    # Works for both Teachers and Students
    member.tell()
```

```
$ python oop_subclass.py
(Initialized SchoolMember: Mrs. Shrividya)
(Initialized Teacher: Mrs. Shrividya)
(Initialized SchoolMember: Swaroop)
(Initialized Student: Swaroop)

Name:"Mrs. Shrividya" Age:"40" Salary: "30000"
Name:"Swaroop" Age:"25" Marks: "75"
```

Class Variables **Object Variables**

All class members are public. One exception: If you use data members with names using the *double underscore prefix* such as `__privatevar`, Python uses name-mangling to effectively make it a private variable.

Thus, the convention followed is that any variable that is to be used only within the class or object should begin with an underscore and all other names are public and can be used by other classes/objects. Remember that this is only a convention and is not enforced by Python (except for the double underscore prefix).

Note for C++/Java/C# Programmers

All class members (including the data members) are *public* and all the methods are *virtual* in Python.

oop_objvar.py

```python
class Robot:
    """Represents a robot, with a name."""

    # A class variable, counting the number of robots
    population = 0

    def __init__(self, name):
        """Initializes the data."""
        self.name = name
        print("(Initializing {})".format(self.name))

        # When this person is created, the robot
        # adds to the population
        Robot.population += 1

    def say_hi(self):
        """Greeting by the robot.

        Yeah, they can do that."""
        print("Greetings, my masters call me {}.".format(self.name))

    def die(self):
        """I am dying."""
        print("{} is being destroyed!".format(self.name))

        Robot.population -= 1

        if Robot.population == 0:
            print("{} was the last one.".format(self.name))
        else:
            print("There are still {:d} robots working.".format(
                Robot.population))

    @classmethod
    def how_many(cls):
        """Prints the current population."""
        print("We have {:d} robots.".format(cls.population))
```

```python
droid1 = Robot("R2-D2")
droid1.say_hi()
Robot.how_many()

droid2 = Robot("C-3PO")
droid2.say_hi()
Robot.how_many()

print("\nRobots can do some work here.\n")

print("Robots have finished their work. So let's destroy them.")
droid1.die()
droid2.die()

Robot.how_many()
```

Output:

```
$ python oop_objvar.py
(Initializing R2-D2)
Greetings, my masters call me R2-D2.
We have 1 robots.
(Initializing C-3PO)
Greetings, my masters call me C-3PO.
We have 2 robots.

Robots can do some work here.

Robots have finished their work. So let's destroy them.
R2-D2 is being destroyed!
There are still 1 robots working.
C-3PO is being destroyed!
C-3PO was the last one.
We have 0 robots.
```