CSE216 – Programming Abstractions
Recitation 5

Objectives:

- Understanding Python built-in class attributes
- Understand inheritance, overloading and polymorphism in Python
- Python code fixing, coding and inheritance exercises

Download Recitation5.zip.

Python built-in class attributes

Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute −

- __dict__ − Dictionary containing the class's namespace.
- __doc__ − Class documentation string or none, if undefined.
- __name__ − Class name.
- __module__ − Module name in which the class is defined. This attribute is "__main__" in interactive mode.
- __bases__ − A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

See Employee.py.

Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero. An object's reference count changes as the number of aliases that point to it changes. An object's reference count increases when it is assigned a new name or placed in a container (list, tuple, or dictionary). The object's reference count decreases when it's deleted with del, its reference is reassigned, or its reference goes out of scope. When an object's reference count reaches zero, Python collects it automatically.

A class can implement the special method __del__(), called a destructor, that is invoked when the instance is about to be destroyed. This method might be used to clean up any non memory resources used by an instance.

See Point.py.

Inheritance and overloading in Python[1]:

Inheritance refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it inherits is called the base (or parent) class.

[1] See book How to Code in Python by Lisa Tagliaferri.

Function overloading − The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved.

Operator overloading − The assignment of more than one function to a particular operator.

Derived classes are declared much like their parent class; however, a list of base classes to inherit from is given after the class name.

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
   'Optional class documentation string'
   class_suite
```

See following programs:

- **Inheritance.py**
- **fish.py**
- **coral_reef.py**

You can always override your parent class methods. One reason for overriding parent's methods is because you may want special or different functionality in your subclass.

See overriding.py

Following are some generic methods that you can override in your own classes.

- __init__ (self [,args...])
  Constructor (with any optional arguments)
  Sample Call : obj = className(args)
- __del__(self)
  Destructor, deletes an object
  Sample Call : del obj
- __repr__(self) **(See Person.py)**
  Evaluable string representation
  Sample Call : repr(obj)
- __str__(self)
  Printable string representation
  Sample Call : str(obj)
- __cmp__ (self, x)
  Object comparison
  Sample Call : cmp(obj, x)

Suppose you have created a Vector class to represent two-dimensional vectors, what happens when you use the plus operator to add them? You could define the __add__ method in your class to perform vector addition and then the plus operator would behave as per expectation.

See Vector.py.

Polymorphism in Python[2]:

Polymorphism is an important feature of class definition in Python that is utilized when you have commonly named methods across classes or subclasses. Polymorphism can be carried out through inheritance, with subclasses making use of base class methods or overriding them.

Python's **duck typing**, a special case of dynamic typing, uses techniques characteristic of polymorphism, including late binding and dynamic dispatch. The term "duck typing" is derived from a quote of writer James Whitcomb Riley: "When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck."

The use of duck typing is concerned with establishing the suitability of an object for a specific purpose. When using normal typing this suitability is determined by the type of an object alone, but with duck typing the presence of methods and properties are used to determine suitability rather than the actual type of the object in question. That is to say, you check whether the object quacks like a duck and walks like a duck rather than asking whether the object is a duck.

When several classes or subclasses have the same method names, but different implementations for these same methods, the classes are polymorphic because they are using a single interface to use with entities of different types. A function will be able to evaluate these polymorphic methods without knowing which classes are invoked.

See polymorphic_fish.py

Exercises:

1. Open file Person1.py.

   Fix the code such that the output is:

   "Marge Simpson"

   "Homer Simpson, 1007"

   Submit corrected Person1.py on blackboard.

2. Open file Person2.py.
   What is the output of the code and why?

3. Taking code from question 2 as reference. Keep the Person class as is. Create two classes namely, SportsPerson and Athlete. Athlete class inherits both, SportsPerson and Person. SportsPerson class has an attribute - isathlete (True or False). Make the user pass person name and a boolean from Command line (isathlete) and write a display() function in

[2] See book How to Code in Python by Lisa Tagliaferri.

Athlete class to print Name, idnumber, and, isathlete. Test your changes with a sample input.

4. Python Object Oriented Programming: Implement a Python class clock. The class Clock simulates the tick-tack of a clock. An instance of this class contains the time, which is stored in the attributes self.hours, self.minutes and self.seconds. Complete the following methods of class clock and submit clock.py file on blackboard:

```
def __init__(self, hours, minutes, seconds):
"""
The parameters hours, minutes and seconds have to be integers and must
satisfy the following equations: 0 <= h < 24, 0 <= m < 60, 0 <= s < 60. An
exception is thrown if the values are outside range.
"""


def tick(self):
"""
This method lets the clock "tick", this means that the internal time will
be advanced by one second.
"""


def __str__(self):
"""
Prints the time in the format HH:MM:SS.
"""
Examples:
>>> x = Clock(12,59,59)
>>> print(x)
   12:59:59
>>> x.tick()
>>> print(x)
   13:00:00
>>> x.tick()
>>> print(x)
   13:00:01
```