Fall 2016

Faculty of Engineering
Computer Engineering Department
CMP 402

Machine Intelligence
Python – Part 3

Objectives:

By the end of this session, students should be able to:

- define a function in python to enhance code modularity.
- know the scope of variables defined
- define and use classes in python.
- know how to use existing python code to solve a problem

Functions

Function definition syntax and scope of variables can be seen in the sample code in Listing 1.1.

Sample code: functions_scopes.py

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:",
```

```
After local assignment: test spam
After nonlocal assignment: nonlocal
spam
After global assignment: nonlocal
spam
In global scope: global spam
```

```
spam)
  do_nonlocal()
  print("After nonlocal
assignment:", spam)
  do_global()
  print("After global assignment:",
spam)

scope_test()
print("In global scope:", spam)
```

Listing 1.1: functions_scopes.py

Classes

- Class definition can be placed at any indentation level. You could conceivably place a class definition in a branch of an if statement, or inside a function. But it should be defined before it is being instantiated.
- Normally class members (including the data members) are public, and all member functions are virtual.
- “Private” instance variables that cannot be accessed except from inside an object don’t exist in Python. However, there is a convention that is followed by most Python code: a name prefixed with an underscore (e.g. _spam) should be treated as a non-public part of the API (whether it is a function, a method or a data member).
- Class *instantiation* uses function notation. When a class defines an *__init__()* method, class instantiation automatically invokes *__init__()* for the newly-created class instance.
- The first argument of a method is called **self**.
- Any function object that is a class attribute defines a method for instances of that class. It is not necessary that the function definition is textually enclosed in the class definition: assigning a function object to a local variable in the class is also ok.

Inheritance

- Classes can inherit from other classes. Built-in types can be used as base classes for extension by the user. The syntax for a derived class definition looks like this:

```
class DerivedClassName(BaseClassName):
```

- Python has two built-in functions that work with inheritance:
    - Use `isinstance()` to check an instance's type: `isinstance(obj, int)` will be `True` only if `obj.__class__` is `int` or some class derived from `int`.
    - Use `issubclass()` to check class inheritance: `issubclass(bool, int)` is `True` since `bool` is a subclass of `int`. However,`issubclass(float, int)` is `False` since `float` is not a subclass of `int`.
    - Python supports a form of multiple inheritance as well. A class definition with multiple base classes looks like this:

```
class DerivedClassName(Base1, Base2, Base3):
```

    - To bundle together a few named data items, you can define an empty class.

```
class Employee:
      pass

john = Employee()  # Create an empty employee record

# Fill the fields of the record
john.name = 'John Doe'
john.dept = 'computer lab'
john.salary = 1000
```

A sample example for class definition and instantiation is shown in Listing 1.2.

```
def compare(self,x1,y1,x2,y2):
    if(x2>x1 and y2>y1):
        return True

class Point:
    def __init__(self,x,y ):
        self.x = x
        self.y = y

class Rectangle:
    """A rectangle class defined by the starting point and the ending point"""
    color="Blue"  # class variable shared by all instances
    f = compare
    def __init__(self, x1, y1,x2,y2):
        if(self.f(x1,y1,x2,y2)):
            self.x1= x1
            self.x2= x2
            self.y1= y1
            self.y2= y2
```

```
    def __initPoints(self,p1,p2):
        self.x1 = p1.x
        self.y1 = p1.y
        self.x2 = p2.x
        self.y2 = p2.y


    def getLength(self):
        return self.x2 - self.x1

    def getWidth(self):
        return self.y2-  self.y1

    def printRectangle(self):
        print("Coordinates of bottom left corner ("+ str(self.x1) + "," +
      str(self.y1)+")")
        print("Coordinates of top right corner (" + str(self.x2) + "," +
      str(self.y2)+")")
        print("Length="+str(self.getLength()))
        print("Width="+str(self.getWidth()))

p1=Point(1,1)
p2=Point(7,5)
rect=Rectangle(p1.x,p1.y,p2.x, p2.y)
rect.printRectangle()
print(rect.__doc__) #prints the documentation line at the class definition
print(Rectangle.color)

for line in open("myfile.txt"):
    print(line, end='')
    coordinates= line.split(" ")
    rect=Rectangle(int(coordinates[0]),int(coordinates[1]),int(coordinates[2]),
int(coordinates[3]))
    rect.printRectangle()
```

Requirement

Write python code to solve the Nqueens problem using genetic algorithm. Use AIMA python code files.
Hints: Using **NqueensProblem** and **GAState** classes and function **genetic_algorithm** in search.py. You will need to:

- Inherit from class GAState to define NqueensGAState and override the mutate function.
- Create random population of size n input from user.
- Define the fitness function
- Call **genetic_algorithm** with your population and your fitness function, change the other parameters and check the effect.

Reference

[1] Python documentation : https://docs.python.org/3/tutorial/classes.html. Last checked at 25 October 2016.