CSC108: Introduction to Computer Programming

Lecture 8

Wael Aboulsaadat

Acknowledgment: these slides are based on material by: Velian Pandeliev, Diane Horton, Michael Samozi, Jennifer Campbell, and Paul Gries from CS UoT

What have we learnt up till now?

- Statements
  - Variables
  - Logical & Mathematical Operators
  - Assignment Statement
  - if/else Statement
  - while loops
- Types & Type conversion
- I/O
  - print
  - input & raw_input
  - Files

What have we learnt up till now?

- Docstrings
- Code Organization
  - Functions
  - Variable scope & Namespaces & Mutability
  - Classes & Objects
  - Inheritance
- Data Structures
  - Lists
  - Tuples
  - Dictionary

Classes & Objects (revisited)

Inheritance

- Recall how Biological inheritance work!
- We have a similar mechanism in Python
- A class can inherit from another!
- What does it mean code-wise?!

Inheritance

- A class can *extend* the definition of another class
  - Allows use (or extension) of methods and attributes already defined in the previous one.
  - New class: *subclass*. Original: *parent*, *ancestor* or *superclass*
- To define a subclass, put the name of the superclass in parentheses after the subclass's name on the first line of the definition.

```
class ChildClass(ParentClass):
```

Definition of a class extending student

```
class Student:
    def __init__(self,Name,Age):
        self.full_name = Name
        self.age = Age
    def get_age(self):
        return self.age
```

```
class Cs_Student (Student):
    def __init__(self,Name,Age,Section):
        Student.__init__(self,Name,Age)
        self.section_num = Section
    def get_section(self):
        return self.section_num
```

Redefining Methods

- To *redefine a method* of the parent class, include a new definition using the same name in the subclass.
  - The old code won't get executed.
- To execute the method in the parent class *in addition to* new code for some method, explicitly call the parent's version of the method.

```
parentClass.methodName(self, a, b, c)
```

  - **The only time you ever explicitly pass 'self' as an argument is when calling a method of an ancestor.**

Extending __init__

- Same as for redefining any other method…
  - Commonly, the ancestor's **__init__** method is executed in addition to new commands.
  - You'll often see something like this in the **__init__** method of subclasses:

```
parentClass.__init__(self, x, y)
```

where parentClass is the name of the parent's class.

Testing

Software Testing

- To test that a function really does what it supposed to be doing!
- You need to know what is the right output to test!
- Call the function, passing it a known input and compare the output !

assert: a python for testing

- Python basic mechanism to test a function

- Syntax

```
assert  Result == Expected, "error message"
```

assert example

```
def power (Num1, Num2):
    return Num1 ** Num2

def testPowerUnitBase():
    result = power(1,5)
    assert result == 1, “1 to power 5 failed”
```

assert example

```
def testPowerUnitBase():
    result = power(1,5)
    assert result == 1, “1 to power 5 failed”
def testPowerPositiveBase():
    result = power(2,6)
    assert result == 64, “2 to 6 failed”
def testPowerNegativeExp():
    result = power(-2,6)
    assert result == -64, “-2 to 6 failed”
def testPowerNegativeExp ():
    result = power(2,-6)
    assert result == 0.015625, “2 to -6 failed”
```

Unit Testing: pyunit

- A python module
- PyUnit is a framework for automating the runnning of test.
- You need to plug your test cases in the framework!
- http://docs.python.org/library/unittest.html

Unit Testing: pyunit - TestCase

- http://docs.python.org/library/unittest.html?highlight=testcase#unittest.TestCase

Unit Testing – how to?

- To use:
  1) Import unittest
  2) For each class write a test class. Test class must inherit from unittest.TestCase
  3) In test class, Write testXYZ() methods using assertEqual, failIf(), failUnless(),….
  4) Override setUp() and tearDown() if necessary
  5) use unittest.main() to run all tests in a file

Example

```
class Employee:
  def __init__(self,Name="",Salary=0.0):
      self.strName = Name
      self.fSalary = Salary

  def  Name(self):
      return self.strName

  def  Salary(self):
      return self.fSalary

  def  addRaise(self,Raise):
      self.fSalary = self.fSalary + Raise
```

Example

```
class EmployeeTestCase(unittest.TestCase):

  # Create employee with non-default values
  def testNonDefaultValues(self):
      b = Employee("Bob", 1500.0)
      self.assertEqual(b.Name(), "Bob")
      self.assertAlmostEqual(b.Salary(), 1500.0, 3)

  # Testing adding raise to salary
  def testAddRaise(self):
      tmp = Employee("Bob", 1500.0)
      tmp.addRaise( 333.5 )
      self.assertAlmostEqual(tmp.Salary(), 1833.5, 3)
```

Sorting

How do you sort a list of numbers?

```
>>> sort([9,1,5,7,8,3,4,6])
[1,3,4,5,6,7,8,9]
```

Bubble sort

```
def bubble_sort(lst):
    bContinue  = True
    while bContinue  == True:
        bContinue = False
        for index in range(len(lst)-1):
            if lst[index] > lst[index+1]:
                temp = lst[index]
                lst[index] = lst[index+1]
                lst[index+1] = temp
                bContinue = True
    return lst

if __name__ == "__main__":
    print bubble_sort([9,1,5,7,8,3,4,6])
```

This Week's To Do List

- Go through lecture slides – make sure you try the code snippets
- Try the lecture's programs posted on course website