Python for Scientific Computing

Lecture 3: Object-oriented Programming

Albert DeFusco
Center for Simulation and Modeling

September 20, 2013

Modules

- Import `math.py` such that `math` becomes the object name

  ```
  import math
   print  math.pi
   print  math.sin(math.pi)
  ```
- Alternatives
  - `from math import sin`
  - `import math as maths`
- Avoid
  - `from math import *`

If you can imagine it, someone probably has a module that can do it.

`http://docs.python.org/2/py-modindex.html`

`http://wiki.python.org/moin/UsefulModules`

Modules

- Any python script can be imported
- The contents are run when imported
- Use __main__ to just import definitions
- Name space defaults to the script's file name

Functions and variables

- Functions can be documented easily

```python
def pi(i):
  """Compute the ith term of the Wallis formula"""
  return 4.*i**2 / (4.*i**2 - 1)

help(pi)
```

- Multiple returns are tuples

```python
def myFunction(x,y):
  return x**2,y*4

a,b = myFunction(y=2,x=8)
```

- Default and optional arguments

```python
def derivative(f,x,h=0.01):
  return (f(x+h) - f(x-h)) / 2.*h

def f(x):
  return x**2

derivative(f,x=0)
```

Functionals and variables

- Functions are objects
- Global variables can be defined
    - Not always good practice
    - May reduce the usability of a module

Name Spaces and Scopes

- Modules
    - Functions

Function Scope

- Variables assigned in a function are private

```
def pi(i):
  """Compute the ith term of the Wallis formula"""
  temp=4.*i**2
  return temp / (temp - 1)

print pi(2)
print temp
```

Function Scope

- Warning!
  - Variables assigned before a function are still in scope
  - It helps to define functions first

```
myVar = 5
def pi(i):
  """Compute the ith term of the Wallis formula"""
  print myVar
  temp=4.*i**2
  return temp / (temp - 1)

print temp
```

Module Scope

- Names assigned in a module are readable by functions
- Names assigned in functions do not affect the outer scope

Object Oriented Programming

- Focus on data, not on the procedure
- Encapsulate procedures with data
- Create *modular* code that can be reused

Object Oriented Programming

- **Class**
  - The description of a *type* of object
- **Object**
  - The realization of the description
  - An *instance* of a class

Object Oriented Programming

- Classes define
  - Attributes
  - Methods
- Instances have
  - data stored in attributes
  - Methods to operate on the data
- Objects can interact with each other by passing attributes to methods

Our modules

```
/home/sam/training/python/lecture3
http://core.sam.pitt.edu/python-fall2013
```

Classes

```python
class shape(object):
  """Shapes have a name and color"""
  def __init__(self,name='shape',color='white'):
    self.name=name
    self.color=color

class Molecule(object):
  """Molecules have a name and chemical formula"""
  def __init__(self,name,formula)
    self.name    = name
    self.formula = formula
```

Operator Overloading

Change or define the behavior of operations

```
class Molecule(object):
...
  def __add__(self, other):
    newName    = self.name + " + " + other.name
    newFormula = "[" + self.formula + "]" + "[" + other.formula +
    return Molecule(newName,newFormula)


mol1=Molecule('water','h2o')
mol2=Molecule('ammonia','nh3')

mol3 = mol1 + mol2
```

Inheritance

- Child classes can be more *specific* than the parent
- Subclasses can override the superclass[†]

```
import math
class shape(object):
  def __init__(self,name='shape',color='white'):
    self.name=name
    self.color=color

class circle(shape):
  def __init__(self,radius=1.,name='circle',color='white'):
  super(circle,self).__init__(name,color)
    self.radius=radius

  def area():
    return math.pi*self.radius**2

class square(shape):
  def __init__(self,size=1.,name='square',color='white'):
  super(square,self).__init__(name,color)
    self.size=size

  def area():
    return self.size**2
```

[†]Polymorphism in Python is achieved when classes implement the same methods, which reduces