Introduction To Python

Week 6: More On Functions And Modules

Dr. Jim Lupo
Asst Dir Computational Enablement
LSU Center for Computation & Technology
Namespace

- Recall some of the import games we played:
  - import pickle
  - import pickle as foofoo
  - from pickle import load
  - from pickle import load as foo
  - from pickle import *
- Names of variables, functions, objects, methods, etc. exist within *namespaces*.
- What a thing is depends on where you are in a code.
Simple Illustration

def myfunc( x ):
    a = 3
    c = x
    d = 0
    print a, c, d

if __name__ == '__main__' :
    a = 4
    c = 13
    d = 42
    myfunc( 17 )
    print a, c, d

What does this print?
Global Namespace

The variables a, c, and d may have the same names, but they are separate entities in each space.
def myfunc( x ):
    global c
    a = 3
    c = x
    d = 0
    print a, c, d

if __name__ == '__main__' :
    a = 4
    c = 13
    d = 42
    myfunc( 17 )
    print a, c, d

Now what does this print?
Values from Different Namespaces

3 17 0
4 17 42

a, d in myfunc() namespace, but c in global namespace.

a, c, d in global namespace

Note: there is also the built-in namespace, which is the namespace reserved for the Python language keywords, functions, etc.
Using **global**

- Some would say NEVER!
- May simplify data handling - i.e. shorter argument lists, dealing with problem setup values, etc.
- May have bad side effects - i.e. new programmer is unaware that other routines may change the value.
- Up to programmer to use it well as the case demands.
Dealing With Errors

• Here is some bad code:

```python
y = 0.0
def myfunc(x):
    return badfunc(x)
def badfunc(x):
    global y
    return x / y

if __name__ == '__main__':
    print myfunc(42.0)
```

• How to anticipate errors and respond?
If It Runs, It Dies!

>>> Traceback (most recent call last):
File "C:/Users/jalupo/Documents/Projects/Presentations/Topical/Python/REU-Course/W6 - 02 Jul/exceptions.py", line 11, in <module>
    print myfunc( 42.0 )
File "C:/Users/jalupo/Documents/Projects/Presentations/Topical/Python/REU-Course/W6 - 02 Jul/exceptions.py", line 4, in myfunc
    return badfunc( x )
File "C:/Users/jalupo/Documents/Projects/Presentations/Topical/Python/REU-Course/W6 - 02 Jul/exceptions.py", line 8, in badfunc
    return x / y
ZeroDivisionError: float division by zero
>>> Nature of the error

Error Location

Was called by

Was called by
Do Error Checking?

- How and where to handle the error?
- Put a conditional check in badfunc()?

```python
def badfunc( x ) :
    global y
    if y != 0.0 :
        return x / y
    return ????
```

- Problem is, what to return, and how to handle in the caller?
Using Error Code or Flag

```python
def badfunc( x ):
    global y
    if y != 0.0:
        return x / y, 0
    else:
        return 0.0, 1
```

What if programmer forgets to check flag?
Check Error Return?

myfunc() would have to look something like this?

```python
def myfunc( x ):
    tmp, ec = badfunc( x )
    if ec:
        return 0.0
    else:
        return tmp
```

Will the caller be happy with 0.0 as an error result?
Exceptions: The Elegant Solution

y = 0

```python
def myfunc(x):
    try:
        return badfunc(x)
    except ZeroDivisionError:
        print "Ooops: Bad Ol' divide-by-zero error!"

def badfunc(x):
    global y
    return x / y

if __name__ == '__main__':
    print myfunc(42.0)
```

Prepare for something bad to happen

Watch for divide by zero exception name.

Action if exception occurs.
Trapping Errors

try alerts Python that errors, called exceptions, may occur (be raised). The syntax is:

```python
try:
    statement block
except exception:
    statement block
```

The try statement block is executed. If an exception is raised, and it matches exception, the except statement block is executed. If not matched, Python will traverse up the call chain until a match is found, or the top level exception handler is reached.
Chose the Proper Level

The example produces output that looks like this:

Ooops: Bad Ol' divide-by-zero error!
None

- Not generally a good idea to have numerical values set to None.
- The goal should be trap exeptions at the level were program can react best (i.e. retry, save data, give other meaningful information).
Move `except` Up a Level

```python
y = 0
def myfunc( x ) :    return badfunc( x )

def badfunc( x ) :    global y    return x / y

if __name__ == '__main__' :    try:        print myfunc( 42.0 )    except ZeroDivisionError:        print "Ooops: Bad Ole' divide-by-zero-error!"
```

Deal With Multiple Errors

```python
try:
    statement block
except ZeroDivisionError:
    statement block
except IOError:
    statement block
except (RuntimeError, TypeError):
    statement block
finally:
    statement block
```

- This allows appropriate actions for each type of error/s.
- **finally** is special - it's statement block is always executed, even if exception is not caught here. It's statement block is executed, then the search for a handler goes up the call the tree.
Legal, But Not Recommended

```python
try:
    statement block
except:
    statement block
```

- Any error is caught
- But, this can disguise a programming error as something else.
Exception Details

• Exceptions represent objects, so you may be able to get more info:

```python
fname = 'foo.txt'
try:
    f = open(fname,'r')
except IOError as e:
    print e.errno
    print fname, ':', e.strerror
```

2
foo.txt : No such file or directory
What Exception To Check For?

- The documentation for most functions includes exceptions that may occur.
- `open()`: If the named file can't be opened, it raises the IOError exception. This fact was used to create the previous example.
- `built-in` exception list. For instance:
  
  https://docs.python.org/2/library/exceptions.html

What exceptions are raised by these other functions we've used: `chr()`, `range()`, `raw_input()`
Raise An Exception

• `raise ExceptionName` allows direct control

```python
y = 0

def myfunc( x ):
    return badfunc( x )

def badfunc( x ):
    global y
    if y == 0.0:
        raise ValueError('Oh, Oh. Y is 0 in badfunc()')
    return x / y

if __name__ == '__main__':
    try:
        print myfunc( 42.0 )
    except ValueError as n:
        print n

# Oh, Oh. Y is 0 in badfunc()
```

Special Case

Use of raise without an exception is allowed in one special case:

```
try:
    statement block
except:
    statement block
raise
```

Do some error handling.

Raises same exception for someone up the call chain to handle.
50,000 Ft Over-flight of Objects

- Encapsulates data and methods.
- An object can inherit from another.
Simple Example: A Point

- A point could have **attributes**:  
  - Coordinates  
  - Mass (physics)  
  - Color  
  - Charge  

- Computed relationships, such as:  
  - Vector between two points  
  - Distance between two points
A Class with Coordinate Attributes

```python
class Point:
    x = 5.
    y = 42.
    z = -10.
p1 = Point
p2 = Point
p1.x = 7.
print p1.x, p2.x
```

Creates a class object named Point

p1 and p1 are bound to same object

Technically, **p1** and **p2** are bound to Point - sort of aliases
Power of **Instantiation**

```python
class Point:
    x = 5.
    y = 42.
    z = -10.

p1 = Point()
p2 = Point()
p1.x = 7.
print p1.x, p2.x
```

- A class object named `Point`
- `p1` and `p1` instantiate `Point` objects
- `7.0 5.0`

`x`, `y`, and `z` become **attributes** of the class `Point` instances.
`p1` and `p2` - unique objects.
Define A Class Method

def getCoords( self ) :
    return self.x, self.y, self.z

The self keyword refers to the object instance.
Example

class Point:
    x = 5.0
    y = 42.0
    z = -10.0

    def getCoord(self):
        return self.x, self.y, self.z

if __name__ == '__main__':
    p1 = Point()
    p2 = Point()
    p1.x = 7.0
    print p1.getCoord()
    print p2.getCoord()
Initializer

class Point :

    def __init__( self, x = 0., y = 0., z = 0 ):
        self.x = x
        self.y = y
        self.z = z

    def getCoord ( self ):
        return self.x, self.y, self.z

if __name__ == '__main__' :
    p1 = Point()
    p2 = Point(10.,20.,30.)
    print p1.getCoord()  
    print p2.getCoord()

(0.0, 0.0, 0.0)
(10.0, 20.0, 30.0)
from math import sqrt

class Point:
    def __init__(self, x = 0., y = 0., z = 0.):
        self.x = x
        self.y = y
        self.z = z

    def getCoord(self):
        return self.x, self.y, self.z

    def distance(self, other):
        x, y, z = other.getCoord()
        d2 = (self.x - x)**2 + (self.y - y)**2 + (self.z - z)**2
        return sqrt(d2)

if __name__ == '__main__':
    p1 = Point()
    p2 = Point(10.0, 20., 30.)
    print p1.distance(p2)
    # 37.4165738677
class Point:
    import math
    def __init__(self, x=0., y=0., z=0.):
        self.x = x
        self.y = y
        self.z = z
    def setCoords(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z
    def getCoords(self):
        return self.x, self.y, self.z
    def distance(self, other):
        x, y, z = other.getCoords()
        d2 = (self.x - x)**2 + (self.y - y)**2 + (self.z - z)**2
        return self.math.sqrt(d2)
    def vector(self, other):
        return (other.getX() - self.x, other.getY() - self.y, other.getZ() - self.z)

if __name__ == '__main__':
    p1 = Point(1., 2., 3.)
    p2 = Point()
    p3 = Point(4., 5., 6.)
    print p1.getCoords()
    print p2.getCoords()
    print p3.getCoords()
    print p2.distance(p3)
    print p1.vector(p3)
    print p3.vector(p1)

(1.0, 2.0, 3.0)
(0.0, 0.0, 0.0)
(4.0, 5.0, 6.0)
8.77496438739
(3.0, 3.0, 3.0)
(-3.0, -3.0, -3.0)
Add Methods

- Vector Dot Product
- Vector Cross Product
Inheritance

class Point:
   x = 5.0
   y = 42.0
   z = -10.0

   def getCoord(self):
      return self.x, self.y, self.z

class Meteor(Point):
   vx = 1.e+5
   vy = -80.0
   vz = 250.

   def getVelocity(self):
      return self.vx, self.vy, self.vz

if __name__ == '__main__':
   m1 = Meteor()
   print m1.getCoord()
   print m1.getVelocity()

(0.0, 0.0, 0.0)
(100000.0, -80.0, 250.0)
What Next?

- We've covered the basic concepts, and enough tools to write a useful program, or read an existing one.
- Go over some useful standard modules?
  - The module: os, sys
  - Using python as a shell scripting tool?
- Adding extension modules?
  - numpy, scipy, biopython, graphics
- Serious program development?
  - Conway Game of Life - cellular automata
  - Laplace heat equation solver - PDE solver
- Declare victory and leave the field?