Modules

Modules are Fundamental

- Modules are programming units that (should) consist of *related* variables and procedures that form a cohesive block of functionality.
- No choice in Python – every file you write is a module.
- Modules allow you to organize your code into logically-connected units. It is a form of *object oriented programming*.
- Modules should contain coherent *data+procedures.*
- Modules permit *data hiding*. Variables and subprograms may be kept private from other program units. This prevents another source of error, by reducing the number of variables an outside program can affect or procedures it can call.

Why Use Modules

- No choice in Python – every file you write is a module.
- Modules allow you to organize your code into logically-connected units. It is a form of *object oriented programming*.
- Modules should contain coherent *data +procedures.*
- Modules permit *data hiding*. Variables and subprograms may be kept private from other program units. This prevents another source of error, by reducing the number of variables an outside program can affect or procedures it can call.

Namespace

- An environment that holds a group of identifiers (variable names, function names, and so forth).
- C, Fortran < 2003 standard: all one big happy *global namespace*. No special identifier.
- Python, etc.: namespaces take the name of the module and/or class in which they are defined.

Python Modules

- Every script file can be/is a module. The file extension must be .py (regardless of operating system).
- Modules are brought into the current namespace with `import`

```
import math
import os
import numpy
```

Imported Module

- When we import the module we must refer to it with its native namespace.

  math.sqrt

  os.getenv

  numpy.zeros (200)

Python Modules (cont.)

- Variations of import
- `from modulename import func1, func2`
  - Now we do not need to precede the function names with the module name.
  - Only func1, func2 can be used.
- `from modulename import *`
  - All function names that do not begin with an underscore (_) are accessible without being preceded by the module name.

Python Modules (more)

- One can rename individual symbols

```
from math import sqrt as squareroot
w=squareroot(10.)
```

- Or one can change the name of the namespace

```
import numpy as np
z=np.zeros(200)
```

Python Main Modules

- Your module can be imported into the interpreter or into another module.
- It will execute everything in the module including requests for input and the like unless you use the special variables __name__ and __main__ (two underscores).
- If you use __main__ you need a `main` function.

Example

```
def main(argv=None):
    if argv is None:
        argv = sys.argv
        x1   = argv[1]

    print "%s%s" % ("x".center(20),"rel error".rjust(12))

    N=6
    for i in range(-N,N+1):
       x=10.0**(-i)
       rootx=sqrt(x)
       relerr=abs((MySqrt(x)-rootx)/rootx)
       print "%14.3e    %15.7e" % (x, relerr)

if __name__=="__main__":
    sys.exit(main())
```

Python Matplotlib

- Matplotlib lets you make very pretty graphics using a syntax similar to Matlab's.
- http://matplotlib.sourceforge.net/
- Includes some of the basic functionality of NumPy so it is not always necessary to import numpy just to make a plot.

Matplotlib Example

```
From pylab import *
N = 30
x = 0.9*rand(N)
y = 0.9*rand(N)
area = pi*(10 * rand(N))**2 # 0 to 10 point radiuses
scatter(x,y,s=area, marker=’o', c='r')
show()
```

NumPy Tutorial

http://www.scipy.org/Tentative_NumPy_Tutorial

Note that the linspace syntax is different from Matlab's. Numpy's is start, stop (for real), and number of divisions of the interval, *not* the increment.