SEE1002
Introduction to Computing for Energy and Environment

Part 3: Basic Python programming

Sec. 1: Modules

Course Outline

Part 1: Introduction to computing

Part 2: Elements of Python programming

Section 1: Data and variables

Section 2: Elementary data structures

Section 3: Branching or decision making

Section 4: Loops

Section 5: Functions

Part 3: Basic Python programming

Section 1: Modules

Section 2: Structure of a Python program

Section 3: Good programming practices

Part 4: Python for science and engineering

Section 1: File input and output

Section 2: Other topics

Outline

1. Motivation
2. Reusing code more efficiently
3. Importing code
4. Referencing functions

1. Motivation

Status

- We have learned enough Python to write fairly sophisticated programs.
- But we haven't talked much about programs as a whole.
- We also want to talk about more advanced features.

Scientific programming versus computer science

- If you were computer science students, you would need to learn many more concepts:
  - More sophisticated data structures
  - Algorithms
  - Different programming models (i.e. alternatives to structured programming)
  - How to design compilers, operating systems and programming languages
- For engineering students, however, there isn't much more to learn. *We have already covered the key concepts!*

2. Reusing code more efficiently

Multiple functions

Recall that we can use multiple functions within a single program.

The order in which the functions are defined doesn't matter.

Example 1a: multiple functions

```
def perimeterareaRectangle(L,W):
    return L*W,2*(L+W)


def perimeterRectangle(L,W):
    return (2*L + 2*W)

def areaRectangle(L,W):
    area=L*W
    return area


L=float(input('Enter length: '))
W=float(input('Enter width: '))

print( 'The area of the rectangle=',areaRectangle(L,W) )
print( 'The perimeter of the rectangle=',perimeterRectangle(L,W) )

area,perimeter=perimeterareaRectangle(L,W)
print( 'The area of the rectangle=',area )
print( 'The perimeter of the rectangle=',perimeter )
```

```
Enter length: 1.0

Enter width: 2.0
The area of the rectangle= 2.0
The perimeter of the rectangle= 6.0
The area of the rectangle= 2.0
The perimeter of the rectangle= 6.0
```

Example 1b: different order

```
def perimeterRectangle(L,W):
    return (2*L + 2*W)

def areaRectangle(L,W):
    area=L*W
    return area

def perimeterareaRectangle(L,W):
    return L*W,2*(L+W)
```

```
L=float(input('Enter length: '))
W=float(input('Enter width: '))

print('The area of the rectangle=',areaRectangle(L,W))
print('The perimeter of the rectangle=',perimeterRectangle(L,W))

area,perimeter=perimeterareaRectangle(L,W)
print( 'The area of the rectangle=',area )
print( 'The perimeter of the rectangle=',perimeter )
```

```
Enter length: 1.0

Enter width: 2.0
The area of the rectangle= 2.0
The perimeter of the rectangle= 6.0
The area of the rectangle= 2.0
The perimeter of the rectangle= 6.0
```

order doesn't matter

Reusing functions

After you've programmed for a while, you'll notice that some functions you've written can be used by different programs. ***How can they be reused?***

- Much of the time, one can just copy and paste the function(s).
- For bigger projects, separate files can be used.

Separate files

If you have lots of functions they can be placed in a separate file. In computer programming, a collection of functions is referred to as a library.

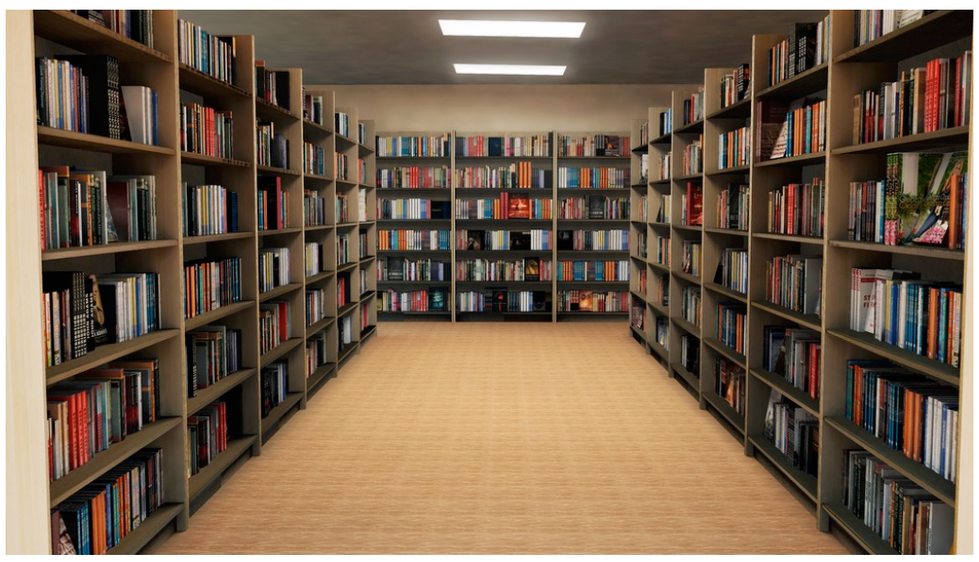
In order to use **all** the functions in a library in Python, the following statement needs to be added to the beginning of the program:

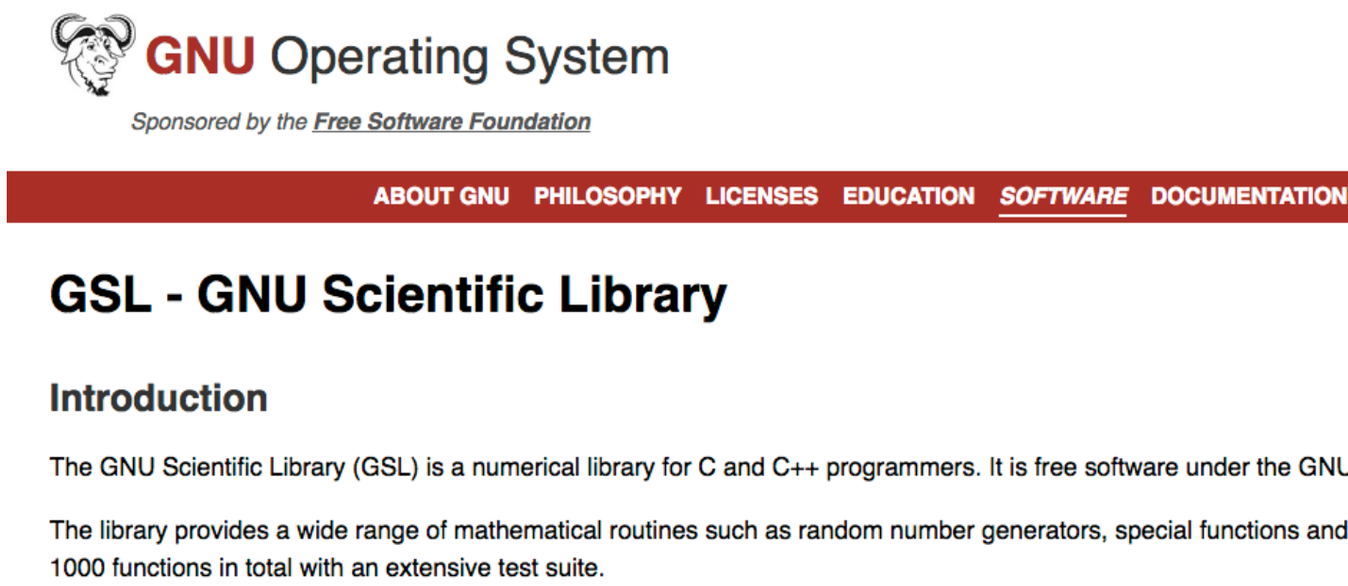
```
from <libname> import *
```

What is a library?

In real life, a library is a collection of information stored in books or an online database.

In computer programming, a library is a collection of functions or subroutines. Specialized libraries exist for many different applications (e.g. graphics, mathematics, handling files, etc.).





Example 2: using a library

```
from mylib import *

L=float(input('Enter length: '))
W=float(input('Enter width: '))

print( 'The area of the rectangle=',areaRectangle(L,W) )
print('The perimeter of the rectangle=',perimeterRectangle(L,W) )

area,perimeter=perimeterareaRectangle(L,W)
print('The area of the rectangle=',area )
print('The perimeter of the rectangle=',perimeter)
```

main program
(example2.py)

```
def perimeterareaRectangle(L,W):
    return (L*W,2*(L+W))

def perimeterRectangle(L,W):
    return (2*L + 2*W)

def areaRectangle(L,W):
    area=L*W
    return (area)
```

functions
(mylib.py)

```
Enter length: 1.0

Enter width: 2.0
The area of the rectangle= 2.0
The perimeter of the rectangle= 6.0
The area of the rectangle= 2.0
The perimeter of the rectangle= 6.0
```

Putting library files in another directory

- In the previous example, the library file is placed in the same directory as the program.
- This is not very convenient if many programs or users need to use the same file.
- To avoid this problem, libraries are often stored in a special location.
- We're not going to cover this because it's not necessary for our purposes. However, the idea is simple: all we need to do is tell Python where to find the file.

3. Importing code

import

- The `import` command allows us to access a module.
- We have already used it on several occasions
- Example:

```
from mylib import *

L=float(input('Enter length: '))
W=float(input('Enter width: '))

print( 'The area of the rectangle=',areaRectangle(L,W) )
print('The perimeter of the rectangle=',perimeterRectangle(L,W) )

area,perimeter=perimeterareaRectangle(L,W)
print('The area of the rectangle=',area )
print('The perimeter of the rectangle=',perimeter)
```

What is a module?

In Python we refer to modules rather than libraries.

A module is actually more general than a library. It contains:

- Functions
- Data
- Data structures

Naming convention

The way we import a module determines how we refer to items within a module, e.g. a function `func`.

There are two ways of doing this.

```
import random
hidden_number = random.randint(1, 50)
input_number = float(input('Please enter a number:'))
```

module name as prefix

```
from mylib import *

L=float(input('Enter length: '))
W=float(input('Enter width: '))

print( 'The area of the rectangle=',areaRectangle(L,W) )
print('The perimeter of the rectangle=',perimeterRectangle(L,W) )

area,perimeter=perimeterareaRectangle(L,W)
print('The area of the rectangle=',area )
print('The perimeter of the rectangle=',perimeter)
```

no prefix

i) Importing a module

- The simplest way to import a module is

```
import <modulename>
```

- Functions belonging need to a module need to be referenced as `modulename.func()` Strictly speaking, such a function is referred to as a method.

- Excluding the prefix yields an error.

```
import math
print(math.sin(0))
print(sin(0))
```

```
0.0
```

```
print(sin(0))

NameError: name 'sin' is not defined
```

Changing the prefix

- If the original module name is too long we can use an abbreviation:

```
import <modulename> as <abbrev>
```

- Functions belonging need to modulename are then referenced as `abbrev.func()`

```
In [122]: import math as m

In [123]: print(m.sin(0))
0.0

In [124]: print(math.sin(0))
Traceback (most recent call last):

  File "<ipython-input-124-a1f67cef7a9a>", line 1, in <module>
    print(math.sin(0))

NameError: name 'math' is not defined
```

ii) Importing a function

- If we import the function explicitly, then we don't need to use the prefix.
- Typically this is done using

```
from <modulename> import *
```

this imports all of the functions **from** the module.

```
from math import *
print(sin(0))
print(cos(0))
```

Useful modules

- Important modules include `math, sys, time, random, numpy` and `scipy.` We will cover them as necessary.

- You can list the contents of a module using `dir(<modulename>)`

4. Referencing functions

Qualification

- A reference of the form `modulename.func()` is referred to as a qualified reference.
- One can make unqualified references using

```
from <modulename> import *
```

  - The meaning of this statement is that **each element in the module can be referenced without a qualifier.**

- Alternatively one can import only the functions one needs:

```
from <modulename> import <functions>
```

Importing multiple functions

- We can import as many functions as we like!
- In practice, however, it doesn't make sense to do so for more than a few functions.

```
9   from math import pi,sin,cos
10  print(sin(pi/2))
11  print(cos(pi/2))
12  print(tan(pi/2))
```

```
    print(tan(pi/2))

NameError: name 'tan' is not defined
```

Use of modules in the shell

- In Spyder (but not in Jupyter notebook), we do not need to import standard modules (e.g. `math`). Thus `pi` and sin() work right away.
- This saves a bit of typing!
- But it's important to keep in mind that
  1. This won't work for all modules (e.g. personal libraries).
  2. This doesn't work for programs!

Why are qualified references useful?

- It seems as though qualified references aren't very useful. Why would we want to use `math.sin()` and `math.pi` instead of `sin()` and `pi`?

  1. *Complicated programs use many modules.* Some of them may have elements with the same name. Qualification avoids conflicts.

  2. *It's a Python standard.* Everybody else does it!

Summary

1. Modules allows us to reuse code that was originally written for another purpose.
2. Before a module can be used, it must be imported.
3. Functions belonging to a module can be called using qualified or unqualified references.