A Modular Approach to Program Organization

Remember that detailing the problem and writing the problem specification are vital to the successful implementation of any potential automated business/scientific process.
In practice, arriving at a well-defined problem specification is extremely important, and very much possible that automated business/Scientific project or processes have failed in implementation due to an inadequate problem specification. I am aware that the problem of today is easy, but we have to learn the right procedure, and prepare you for more sophisticated problem in the future.

A Modular Approach to Program Organization

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't

What is module

A *module* is a collection of variables and functions that are grouped together in a single file.

The variables and functions in a module are usually related to one another in some way;
for example, module math contains the variable pi and mathematical functions such as cos (cosine) and sqrt (square root).

We will show you how to use some of the hundreds of modules that come with Python

How to create your own modules.

Importing Modules

To gain access to the variables and functions from a module, you have to *import* it. To tell Python that you want to use functions in module math, for example, you use this import statement:

```
>>> import math
```

Importing a module creates a new variable with that name. That variable refers to an object whose type is module:

```
>>> type(math)
<class 'module'>
```

Importing Modules

Once you have imported a module, you can use built-in function help to see what it contains. Here is the first part of the help output:

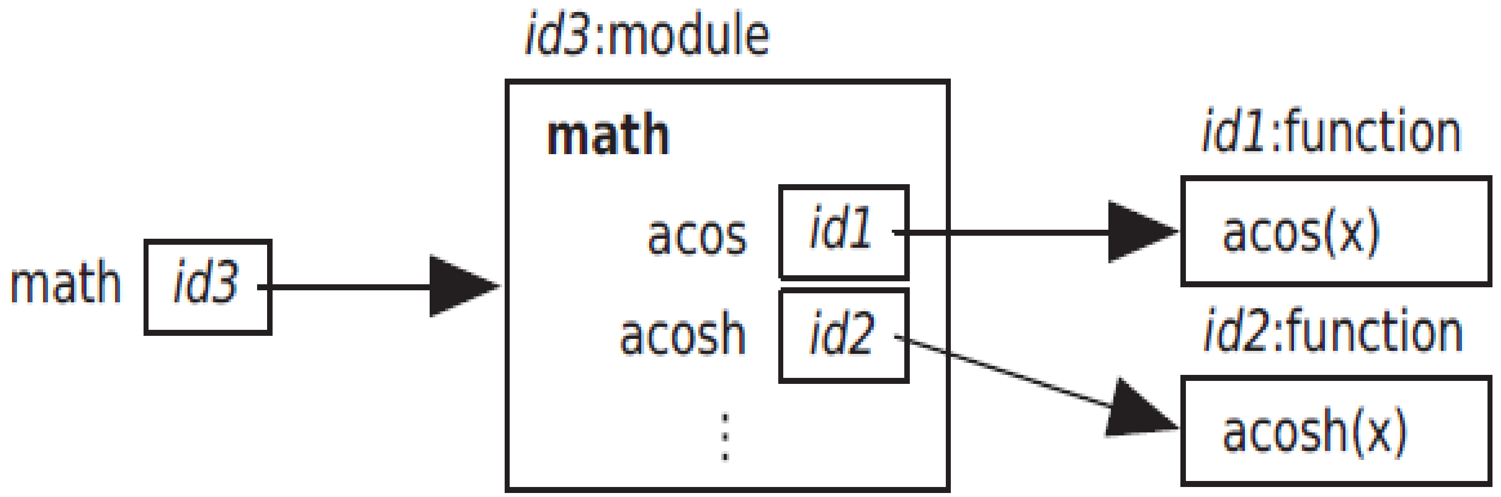
>>> help(math)

MODULE REFERENCE:

http://docs.python.org/3.3/library/math

Math module provides access to the 100's of mathematical functions
Ex:
acos(...)
acos(x)
acosh(...)
acosh(x)
Return the hyperbolic arc cosine (measured in radians) of x.
[Lots of other functions not shown here.]

Importing Modules



Ex: is this the correct reference?

```
>>>import math
>>> sqrt(9)                ?
```

Try it

Is this correct?

Importing Modules

Modules can contain more than just functions.

Module math, for example, also defines some variables like pi.

Once the module has been imported, you can use these variables like any others:

```
>>> import math
>>> math.pi
3.141592653589793
>>> radius = 5
>>> print('area is', math.pi * radius ** 2)
area is 78.53981633974483
```

Importing Modules

You can even customize any variables imported from modules to your own :

```
>>> import math
>>> math.pi = 3
>>> radius = 5
>>> print('area is', math.pi * radius ** 2)

area is 75
```

Don’t do this! Changing the value of π isn’t a good idea. BUT , you might need to customize other scientific constants according to your case problem

Importing Modules

Python lets you specify exactly what you want to import from a module, like this:

```
>>> from math import sqrt, pi
>>> sqrt(9)
?
>>> radius = 5
>>> print('circumference is', 2 * pi * radius)
circumference is 31.41592653589793
```

This doesn’t introduce a variable called math. Instead, it creates function sqrt and variable pi in the current namespace, as if you had typed the function definition and variable assignment yourself.

Restoring a Module

Without having to restart the shell, you can restore the module to its original state using function reload from module imp:

```
>>> import math
>>> math.pi = 3
>>> math.pi
3
>>> import imp
>>> math = imp.reload(math)
>>> math.pi
3.141592653589793
>>>
```

Function imp.reload returns the original non updated module.

Important note

If you import a function called spell from a module called
magic and then you import another function called spell from the grammar module, the second replaces the first. It's exactly like assigning one value to a variable and then assigning another value: the most recent assignment or import wins.
This is why it's usually *not* a good idea to use import *, which brings in everything from the module at once:

```
>>> from math import *
>>> print(sqrt(8))
2.8284271247461903
```

Although import * saves some typing, you run the risk of your program accessing the incorrect function and not working properly.

Also, you upload your memory with all functions

Module __builtins__

Python’s built-in functions are actually in a module named __builtins__

(with two underscores before and after ’builtins’).

You can see what’s in the module using:

```
>>>help(__builtins__)
```

or if you just want to see what functions and variables are available, you can use:

```
>>> dir(__builtins__)
```

Defining Your Own Modules

Let us reuse some function we have already created and create a customized module out of them:

How to create module with many function (each function is accessible via the module name ‘period’ and then the function.

Design the function definition for convert_to_celsius

Save in a file called temperature.py. (You can save this file anywhere you like, although most programmers create a separate directory for each set of related files that they write.)

Now add another function to temperature.py called above_freezing that returns True if and only if its parameter celsius is above freezing:

Defining Your Own Modules

```python
def convert_to_celsius(fahrenheit):
    """ (number) -> float

    Return the number of Celsius degrees equivalent to fahrenheit degrees.

    >>> convert_to_celsius(75)
    23.88888888888889
    """

    return (fahrenheit - 32.0) * 5.0 / 9.0


def above_freezing(celsius):
    """ (number) -> bool

    Return True iff temperature     celsius degrees is above freezing.

    >>> above_freezing(5.2)
    True
    >>> above_freezing(-2)
    False
    """

    return celsius > 0
```

Defining Your Own Modules

By saving the whole functions in a file temperature.py; you have created a module called temperature. Now that you've created this file, you can run it and import it like any other module:

```
>>> import temperature
>>> celsius = temperature.convert_to_celsius(33.3)
>>> temperature.above_freezing(celsius)
True
```

What Happens During Import

Let us run this module

```
def experiment():
     return ("your prof name is Sadiq")
```

```
>>> import experiment
>>> experiment.experiment()
```

Your prof name is Sadiq

Let us edit the module to:

```
def experiment():

   return ("your prof name is Vida")
```

```
>>> experiment.experiment()
```

What shall it give?

In Class Exercise

Create module calculator, which include the following function:

1. Function return True /False for even/odd, the call could be:
   is_Even(number) -> bool
2. Function get the integer part of a number, the call could be:
   Int_part(float) -> int
3. Function check_range which check umber input to the function and see if the number in the range 100 – 200 (inclusive) or not, the call could be check_range(number) -> string (i.e:
“Number with in the range” or “Number out of the range”
4. Give prompt statement to each function:
   example: Please give the number.

Write your code in a file, save it exercise7.py , run it and execute from the python shell.