Modules

We can define functions and have them do whatever we want. The logical question to ask is "Do we have to define a function for everything? Like square roots or absolute values?"

No, we have modules!

Modules are collections of functions and values that other people have created that we can use in our code. In other languages, these are called libraries, but we will call them modules because that is what the Python community does.

math

I don't want to through every module under the sun because there are a lot of modules, but I will talk about the math module.

To add a library to your code, you "import the module". Here is the code to import the math module and use it:

```
import math

print math.pi
print math.ceil(4.5)
print math.ceil(4.1)
print math.trunc(math.pi)
print math.sqrt(3)
```

This would display:

```
3.14159265359

5

5

3

1.77245385091
```

If you ever wonder what is inside a module, you can do the following:

```
import math
help(math)
```

The help() function is very useful for these sort of things.

Structure of importing

When you import a module, you must have the following structure:

```
import MODULE_NAME
MODULE_NAME.FUNCTION_NAME(argument)
```

However, writing out `math.pi` instead of `pi` is too much work! Thankfully, Python allows you to go around that. Let's say we only want the value of `pi` from the `math` module and we do not want *all* of the `math` module.

```
from math import pi
```

That's pretty easy. I also want `sqrt()` as well.

```
from math import pi, sqrt
```

Nice.

The extreme case of this is import everything from `math` without import `math`. We can do that.

```
from math import *
print pi
```

```
3.14159265359
```

What we have done, in a way, is import all the functions from the `math` module without importing the `math` module.

What if you don't like the name of the module? We can change the name module when we import it.

```
import math as m
print m.e
```

```
2.718281828459045
```

A warning about importing

When using `as` and `from` for import, you could cause functions from different modules that have the same name to overwrite each other. Here is an example of overwriting a function:

```
def add(a,b):
  return a+b

def add(a,b):
  return "nope"

print add(1,2)
```

This will display `nope`, not `3`. This can happen when you import a couple modules. There is a chance there is some overlap with names so a problem like this could happen. Be careful.

Importing your own modules

Modules aren't special. the `math` module is special because the people who made Python made it, but you could make your own module. Modules are files. Here are two files that I wrote up:

shapes.py

```
from math import pi

def rectArea(height, width):
    return height*width
def rectPara(height,width):
    return 2*(height+width)

def circArea(radius):
    return pi * radius ** 2
def circPara(radius):
    return 2 * radius * pi

def squareArea(side):
    return side ** 2
def squarePara(side):
    return 4 * side
```

example.py

```
import shapes
print shapes.squareArea(2)
```

This will run and display `4`. You can do all the syntax of using `as` and `for` for your own files.