How to Create a Python Module

We will begin by creating a super simple module. This module will provide us with basic arithmetic and no error handling. Here's our first example:

```
def add(x, y):
    return x + y

def division(x, y):
    return x / y

def multiply(x, y):
    return x * y

def subtract(x, y):
    return x - y
```

This code has issues, of course. If you pass in two Integers to the **division** method, then you'll end up getting an Integer back if you are using Python 2.x. This may not be what you're expecting. There's also no error checking for division by zero or mixing of strings and numbers. But that's not the point. The point is that if you save this code, you have a fully qualified module. Let's call it **arithmetic.py**. Now what can you do with a module anyway? You can import it and use any of the defined functions or classes that are inside it. And we could make it **executable** with a little spit and polish. Let's do both!

First we'll write a little script that imports our module and runs the functions in it. Save the following as **math_test.py**:

```
# import arithmetic

print(add(5, 8)) # print(arithmetic.add(5, 8))
print(subtract(10, 5))
print(division(2, 7))
print(multiply(12, 6))
```

Now let’s modify the original script so that we can run it from the command line. Here’s one of the simplest ways to do it:

```
def add(x, y):
    return x + y

def division(x, y):
    return x / y

def multiply(x, y):
    return x * y

def subtract(x, y):
    return x - y

if __name__ == "__main__":
    import sys
    print(sys.argv)
    v = sys.argv[1].lower()
    valOne = int(sys.argv[2])
    valTwo = int(sys.argv[3])
    if v == "a":
        print(add(valOne, valTwo))
    elif v == "d":
        print(division(valOne, valTwo))
    elif v == "m":
        print(multiply(valOne, valTwo))
    elif v == "s":
        print(subtract(valOne, valTwo))
    else:
        pass
```

The proper way to do this script would be to use Python’s **optparse (pre-2.7)** or **argparse (2.7+)** module. You should spend some time to figure out one of these modules as a learning exercise. In the meantime, we will move on to packages!