How to Create a Python Package

WE'LL COVER THE FOLLOWING

- Wrapping Up

The main difference between a module and a package is that a package is a collection of modules AND it has an **__init__.py** file. Depending on the complexity of the package, it may have more than one **__init__.py**. Let's take a look at a simple folder structure to make this more obvious, then we'll create some simple code to follow the structure we define.

```
mymath/
    __init__.py
    adv/
        __init__.py
        sqrt.py
    add.py
    subtract.py
    multiply.py
    divide.py
```

Now we just need to replicate this structure in our own package. Let's give that a whirl! Create each of these files in a folder tree like the above example. For the add, subtract, multiply and divide files, you can use the functions we created in the earlier example. For the **sqrt.py** module, we'll use the following code.

```
# sqrt.py
import math

def squareroot(n):
    return math.sqrt(n)
```

You can leave both **__init__.py** files blank, but then you'll have to write code like **mymath.add.add(x,y)** which is pretty ugly, so we'll add the following

code to the outer **__init__.py** to make using our package easier to understand and use.

```
# outer __init__.py
from . add import add
from . divide import division
from . multiply import multiply
from . subtract import subtract
from .adv.sqrt import squareroot
```

Now we should be able to use our module once we have it on our Python path. You can copy the folder into your Python's **site-packages** folder to do this. On Windows it's in the following general location: **C:\Python34\Lib\site-packages**. Alternatively, you can edit the path on the fly in your test code. Let's see how that's done:

```
import sys

# modify this path to match your environment
sys.path.append('C:\Users\mdriscoll\Documents')

import mymath

print(mymath.add(4,5))
print(mymath.division(4, 2))
print(mymath.multiply(10, 5))
print(mymath.squareroot(48))
```

Note that my path does NOT include the **mymath** folder. You want to append the parent folder that holds your new module, NOT the module folder itself. If you do this, then the code above should work.

You can also create a **setup.py** script and install your package in **develop** mode. Here's an example setup.py script:

```
#!/usr/bin/env python

from setuptools import setup


# This setup is suitable for "python setup.py develop".

setup(name='mymath',
      version='0.1',
      description='A silly math package',
```

```
      author='Mike Driscoll',
      author_email='mike@mymath.org',
      url='http://www.mymath.org/',

      packages=['mymath', 'mymath.adv'],
      )
```

You would save this script one level above the **mymath** folder. To install the package in develop mode, you would do the following:

```
python setup.py develop
```

This will install a link file in the site-packages folder that points to where ever your package resides. This is great for testing without actually installing your package.

Congratulations! You've just created a Python package!

Wrapping Up

You've just learned how to create your very own, custom-made modules and packages. You will find that the more you code, the more often you'll create programs that have parts that you want to re-use. You can put those reusable pieces of code into modules. Eventually you will have enough related modules that you may want to turn them into a package. Now you have the tools to actually do that!