# Command-Line Interface

The unittest module comes with a few other commands that you might find useful. To find out what they are, you can run the unittest module directly and pass it the **-h** as shown below:

```
python -m unittest -h
```

This will cause the following output to be printed to stdout. Note that I have cut out a portion of the output that covered **Test Discovery** command line options for brevity:

```
usage: python -m unittest [-h] [-v] [-q] [--locals] [-f] [-c] [-b]
                          [tests [tests ...]]

positional arguments:
  tests           a list of any number of test modules, classes and test
                  methods.

optional arguments:
  -h, --help      show this help message and exit
  -v, --verbose   Verbose output
  -q, --quiet     Quiet output
  --locals        Show local variables in tracebacks
  -f, --failfast  Stop on first fail or error
  -c, --catch     Catch ctrl-C and display results so far
  -b, --buffer    Buffer stdout and stderr during tests

Examples:
  python -m unittest test_module               - run tests from test_module
  python -m unittest module.TestClass          - run tests from module.TestClass
  python -m unittest module.Class.test_method  - run specified test method
```

Now we have some ideas of how we might call our test code if it didn't have the call to **unittest.main()** at the bottom. In fact, go ahead and re-save that code with a different name, such as **test_mymath2.py** with the last two lines removed. Then run the following command:

```
python -m unittest test_mymath2.py
```

This should result in the same output we got previously:

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

The cool thing about using the unittest module on the command line is that we can use it to call specific functions in our test. Here's an example:

```
python -m unittest test_mymath2.TestAdd.test_add_integers
```

This command will run only run test, so the output from this command should look like this:

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Alternatively, if you had multiple test cases in this test module, then you could call just one test case at a time, like this:

```
python -m unittest test_mymath2.TestAdd
```

All this does is call our **TestAdd** subclass and runs all the test methods in it. So the result should be the same as if we ran it in the first example:

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

The point of this exercise is that if you were to have additional test cases in this test module, then this method gives you a method to run individual test cases instead of all of them.