Testing with doctest

WE'LL COVER THE FOLLOWING

The doctest module will search for pieces of text in your code that resemble interactive Python sessions. It will then execute those sessions to verify that they work exactly as written. This means that if you wrote an example in a docstring that showed the output with a trailing space or tab, then the actual output of the function has to have that trailing whitespace too. Most of the time, the docstring is where you will want to put your tests. The following aspects of doctest will be covered:

> - How to run doctest from the terminal
> - How to use doctest inside a module
> - How to run a doctest from a separate file

Let's get started!

Running doctest via the Terminal

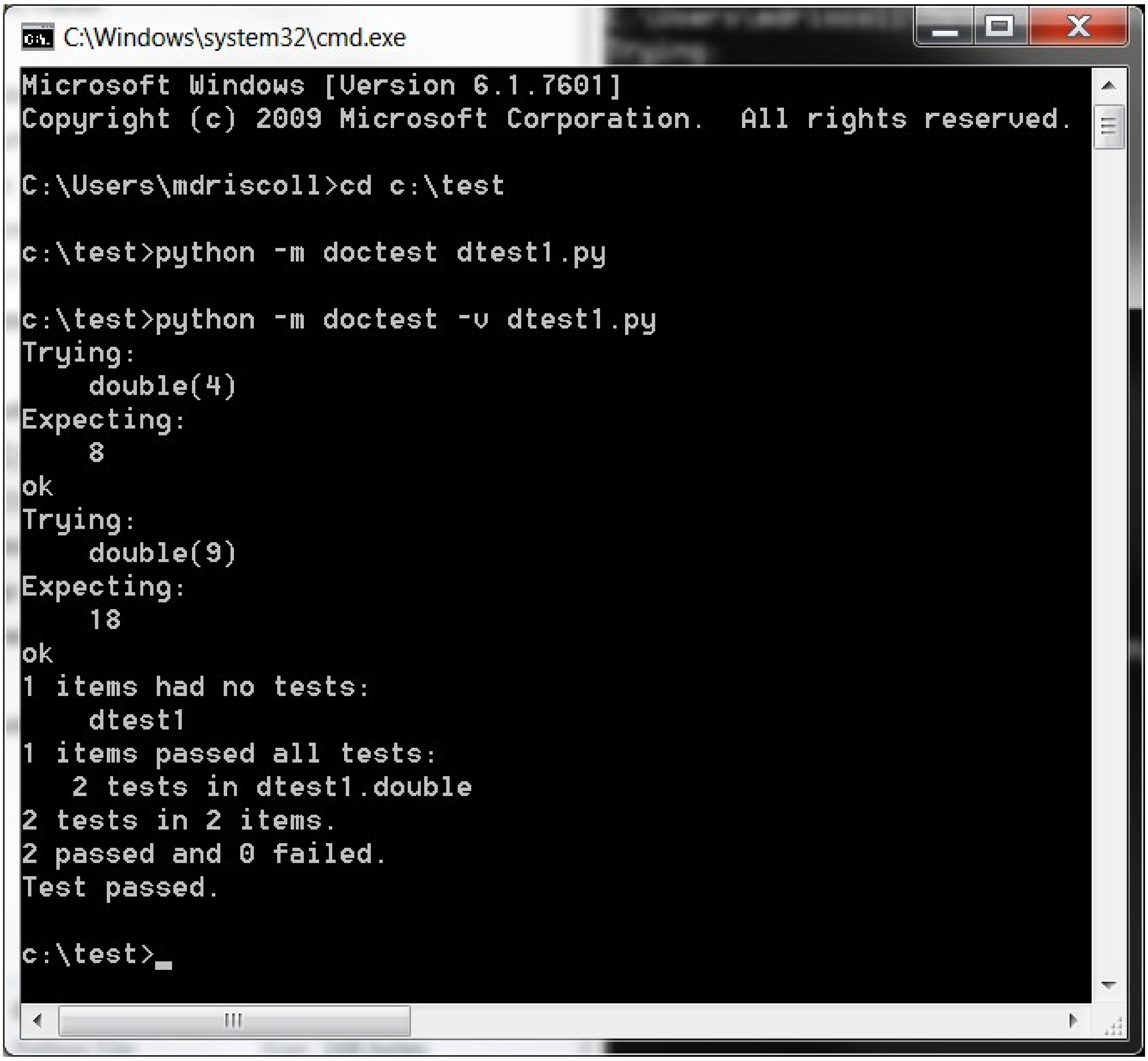
We will start by creating a really simple function that will double whatever is given to it. We will include a couple of tests inside the function's **docstring**. Here's the code (be sure and save it as "dtest1.py"):

```
# dtest1.py

def double(a):
    """
    >>> double(4)
    8
    >>> double(9)

    18
    """
    return a*2
```

Now we just need to run this code in **doctest**. Open up a terminal (or command line) and change directories to the folder that contains your script. Here's a screenshot of what I did:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\mdriscoll>cd c:\test

c:\test>python -m doctest dtest1.py

c:\test>python -m doctest -v dtest1.py
Trying:
    double(4)
Expecting:
    8
ok
Trying:
    double(9)
Expecting:
    18
ok
1 items had no tests:
    dtest1
1 items passed all tests:
   2 tests in dtest1.double
2 tests in 2 items.
2 passed and 0 failed.
Test passed.

c:\test>_
```

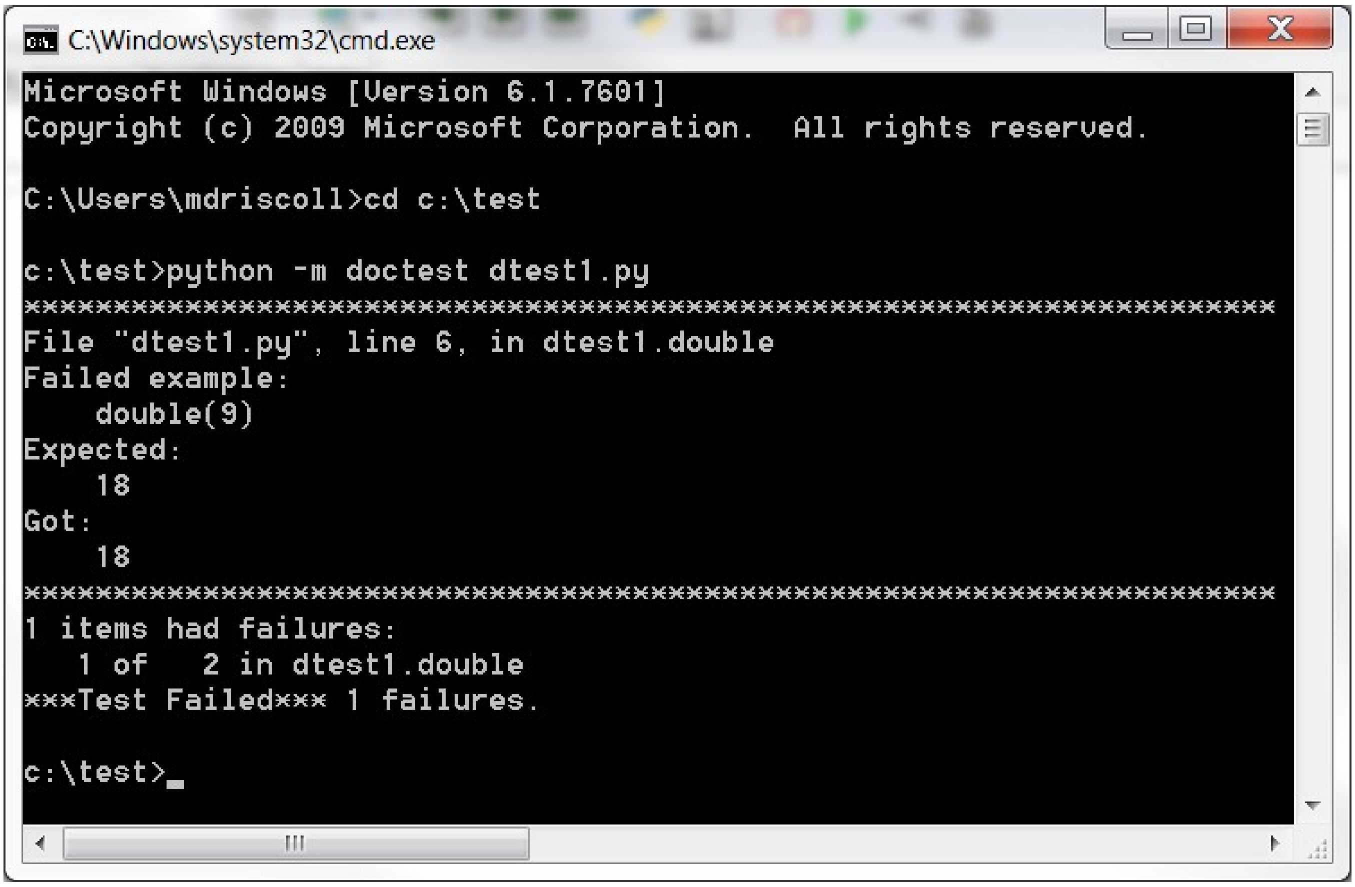
You will notice that in the first example, I executed the following:

```
python -m doctest dtest1.py
```

That ran the test and nothing printed out to the screen. When you don't see anything printed, that means that all the tests passed successfully. The second example shows the following command:

```
python -m doctest -v dtest1.py
```

The “-v” means that we want verbose output, which is exactly what we received. Open up the code again and add a space after the “18” in the docstring. Then re-run the test. Here’s the output I received:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\mdriscoll>cd c:\test

c:\test>python -m doctest dtest1.py
**********************************************************************
File "dtest1.py", line 6, in dtest1.double
Failed example:
    double(9)
Expected:
    18
Got:
    18
**********************************************************************
1 items had failures:
   1 of   2 in dtest1.double
***Test Failed*** 1 failures.

c:\test>_
```

The error message says it expected “18” and it got “18”. What’s going on here? Well we added a space after “18” to our docstring, so doctest actually expected the number “18” followed by a space. Also beware of putting dictionaries as output in your docstring examples. Dictionaries can be in any order, so the likelihood of it matching your actual output isn’t very good.

Running doctest Inside a Module

Let’s modify the example slightly so that we import the **doctest** module and use its **testmod** function.

```
def double(a):
    """
    >>> double(4)
    8
    >>> double(9)
    18
    """
    return a*2

if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose=True)
```

Here we import **doctest** and call **doctest.testmod**. We pass it the keyword argument of **verbose=True** so that we can see some output. Otherwise, this script will run with no output whatsoever, signifying that the tests ran successfully.

If you do not want to hard-code the verbose option, you can also do it on the command line:

```
python dtest2.py -v
```

Now we're ready to learn how to put the tests into a separate file.

Running doctest From a Separate File

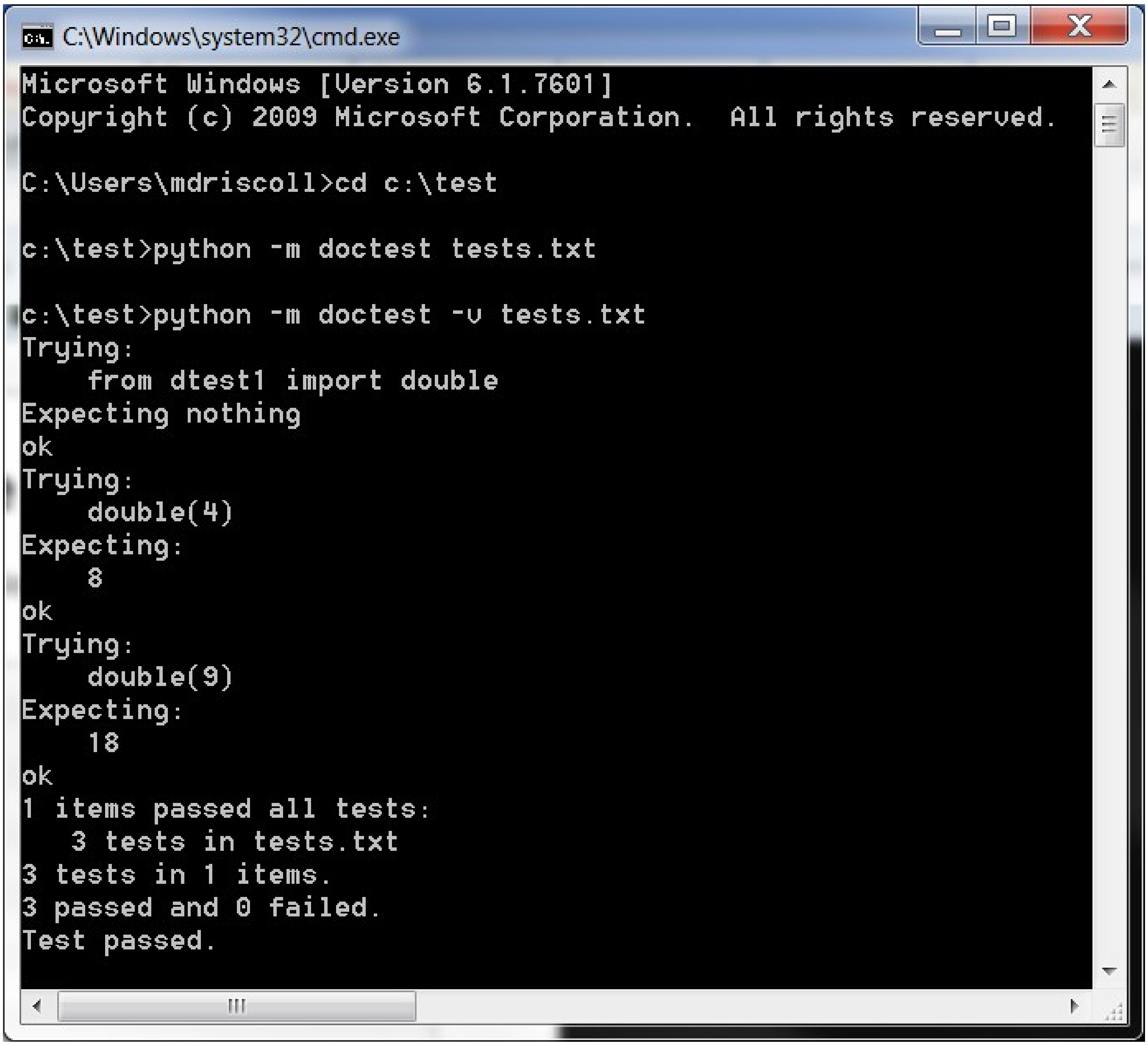
The doctest module also supports putting the testing into a separate file. This allows us to separate the tests from the code. Let's strip the tests from the previous example and put them into a text file named **tests.txt**:

```
#The following are tests for dtest2.py

print(double(4))
#8

print(double(9))
#18
```

Let's run this test file on the command line. Here's how:



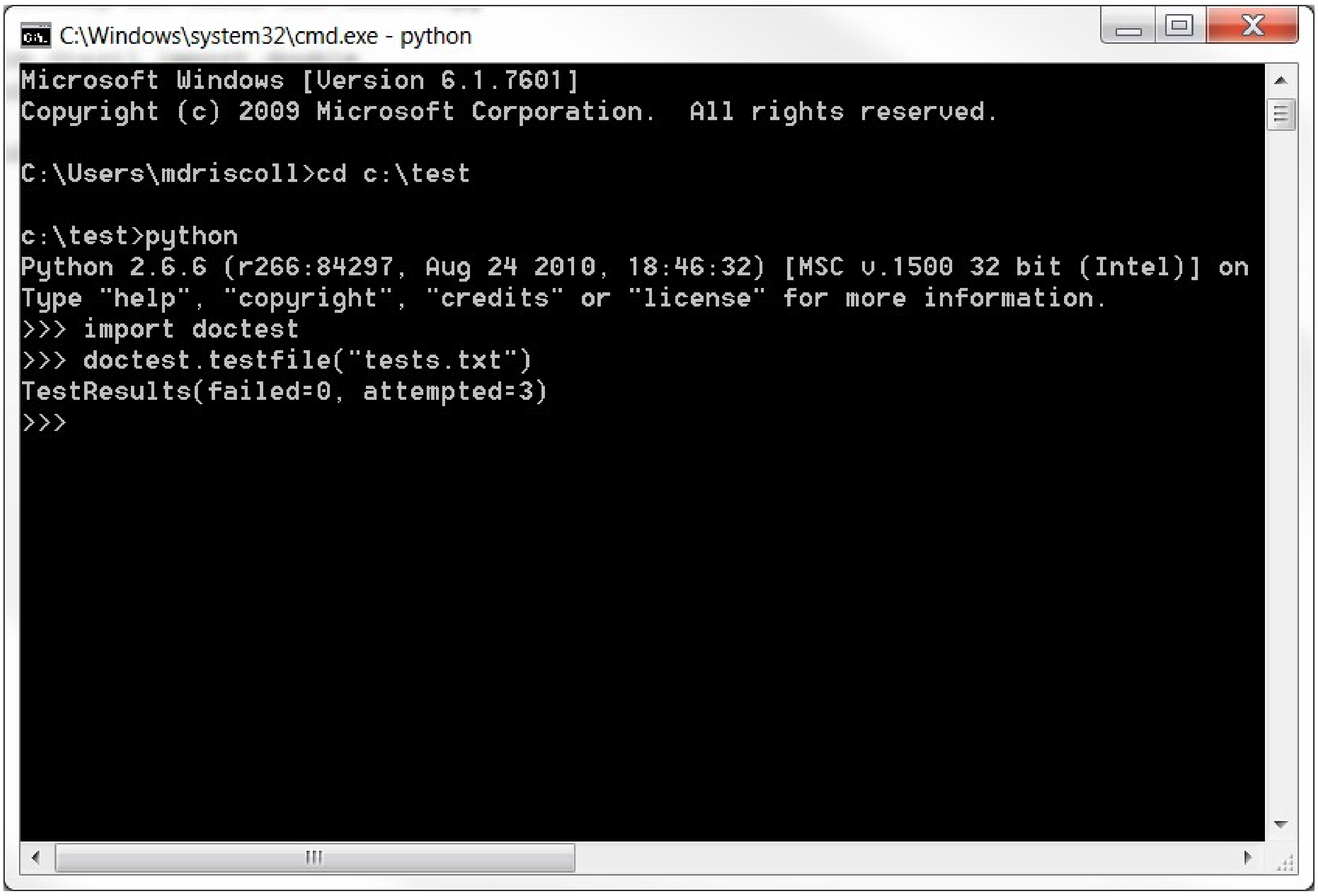
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\mdriscoll>cd c:\test

c:\test>python -m doctest tests.txt

c:\test>python -m doctest -v tests.txt
Trying:
    from dtest1 import double
Expecting nothing
ok
Trying:
    double(4)
Expecting:
    8
ok
Trying:
    double(9)
Expecting:
    18
ok
1 items passed all tests:
   3 tests in tests.txt
3 tests in 1 items.
3 passed and 0 failed.
Test passed.
```

You will notice that the syntax for calling doctest with a text file is the same as calling it with a Python file. The results are the same as well. In this case, there are three tests instead of two because we're also importing a module. You can also run the tests that are in a text file inside the Python interpreter. Here's one example:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\mdriscoll>cd c:\test

c:\test>python
Python 2.6.6 (r266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)] on
Type "help", "copyright", "credits" or "license" for more information.
>>> import doctest
>>> doctest.testfile("tests.txt")
TestResults(failed=0, attempted=3)
>>>
```

Here we just import **doctest** and call its **testfile** method. Note that you need to also pass the filename or path to the **testfile** function. It will return a **TestResults** object that contains how many tests were attempted and how many failed.