Autospeccing

The mock module also supports the concept of **auto-speccing**. The autospec allows you to create mock objects that contain the same attributes and methods of the objects that you are replacing with your mock. They will even have the same call signature as the real object! You can create an autospec with the **create_autospec** function or by passing in the **autospec** argument to the mock library's **patch** decorator, which we'll be looking at in the next section.

For now, let's look at an easy-to-understand example of the autospec:

```
from unittest.mock import create_autospec
def add(a, b):
    return a + b

mocked_func = create_autospec(add, return_value=10)
print (mocked_func(1, 2))
#10

mocked_func(1, 2, 3)
#Traceback (most recent call last):
# File "/usercode/__ed_file.py", line 9, in <module>
# mocked_func(1, 2, 3)
# File "<string>", line 2, in add
# File "/usr/lib/python3.5/unittest/mock.py", line 183, in checksig
# sig.bind(*args, **kwargs)
# File "/usr/lib/python3.5/inspect.py", line 2918, in bind
# return args[0]._bind(args[1:], kwargs)
# File "/usr/lib/python3.5/inspect.py", line 2839, in _bind
# raise TypeError('too many positional arguments') from None
#TypeError: too many positional arguments
```

In this example, we import the **create_autospec** function and then create a simple adding function. Next we use create_autospec() by passing it our **add** function and setting its return value to 10. As long as you pass this new mocked version of add with two arguments, it will always return 10. However, if you call it with the incorrect number of arguments, you will receive an exception.