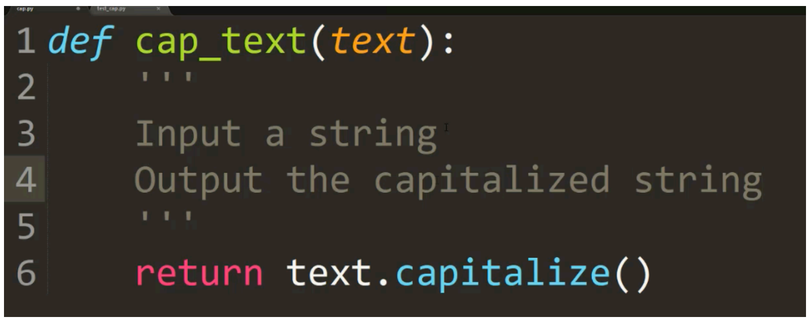
SECTION 10: ERRORS AND EXCEPTIONS HANDLING - 46 minutes, 6 parts
6/6 Running tests with the Unittest Library

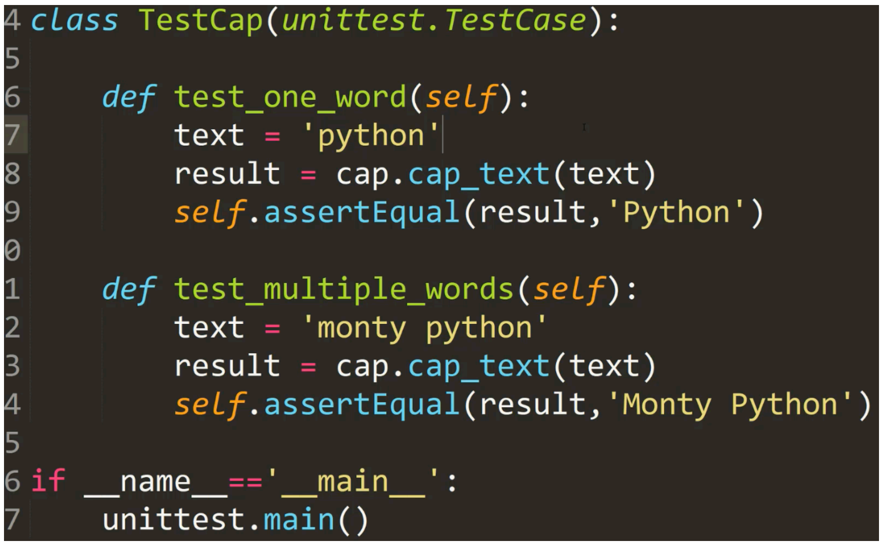
- **Writing our own unit tests for the code**
  - -> writing our own unit tests
  - -> seeing if the program gives us the expected retults
  - **-> we have a .py file with the program, and then we have another .py file which contains the tests for that program**
  - **-> so we have two .py files**



```
def cap_text(text):
    '''
    Input a string
    Output the capitalized string
    '''
    return text.capitalize()
```

- **In the .program .py file**
  - -> he writes a function which returns text
  - -> in this case it's a function which capitalises that text
  - -> he is decorating the code with comments

- **In the test.py file**
  - -> he is importing the unittest module to write a unit test
  - -> and then importing the .name of the .py file which contains the Python for the program
  - -> he defines a class, using unittest
  - -> then defines methods inside this which will be run when we test the script
  - -> he stores a test string in a variable, then runs the function from the imported .py file on the test string
  - -> then uses the .assertEqual method on it
  - -> this is inherited from the unittest module
  - -> he then defines another test function, for a different case on the same piece of code
  - **-> in the test module, we are creating multiple different cases which could be entered into the program**
  - **-> and then the if __name__ == '__main__' block at the end of the .py test file**
  - **-> then he runs the .py test file in the terminal**
  - **-> depending on the outcome of this, the issue can be the test itself, or the script**
  - -> QA people write the tests to make sure that they work as expected
  - -> in this example, we capitliased the first letter of the string
  - -> there is another method called .title
  - **-> self.assertEqual <- run the code from the unit test, and check if the outcome which the program gives us is the expected one**
  - -> then running the test in the terminal
  - -> the problem might also be the test itself
  - -> running the test after changing it, until it works
  - **-> we have a .py file which contains the unit tests, and another which contains the program which it imports in**
  - **-> we are listing out the different possible outcomes for a function**



```
class TestCap(unittest.TestCase):

    def test_one_word(self):
        text = 'python'
        result = cap.cap_text(text)
        self.assertEqual(result,'Python')

    def test_multiple_words(self):
        text = 'monty python'
        result = cap.cap_text(text)
        self.assertEqual(result,'Monty Python')

if __name__=='__main__':
    unittest.main()
```