

Homework 2 Report

Software Testing and QA

Cameron Byrne

CMB1366

GitHub Username: Cammbyrne

Functions

Each requirement, being the Body Mass Index calculator and the Retirement Age calculator, are in separate files, each creating the function for it to be called in the test case file.

The BMI function has the inputs of the users weight, the height in feet, and the height in inches. Each input is constrained to be positive, whole numbers. We are also assuming no user error in inputting them. For example, the user will not type 5" or 5 feet, they will just type 5. This helps justify the boundaries by keeping them limited to 1 decimal place. The function calculates the BMI by using the formula given the users inputs and then determines whether the user is underweight, overweight, or an average weight. It returns those values in the form of a tuple.

The Retirement function has the inputs of the users age, current salary, percent saved to the goal, and the desired savings goal. The constraints for age, salary, and desired goal are being positive, whole numbers. The percent saved is constrained to be a percent in the form of a decimal. The whole system uses the assumption that the user will not live past 100, and also that there will be no user error in the inputs. This allows the formula to correctly give the age the goal is met. The function takes the salary and multiplies it by the percent saved and then multiplies that by 1.35. This is to add the 35% that the company matches on the current saved percent. This gives us the savings per year, which the function will divide the savings goal by to find the amount of years to go. Adding the age and years to go will give the user what age they will meet the goal at. If the age is below 100, the function will tell them that they can meet the goal before they die. If it is 100 or above, it will tell them they cannot meet the goal. It returns these in the form of a tuple

Each test case file uses the unittest library from python. In each, a class is created for the test cases, in which, are the cases themselves.

For the BMI test cases, there are 7 in total:

The first uses the values 180 for the weight, and 5 feet and 7 inches for the height

The expected outcome is a BMI of 28.9 and a return of Average Weight

The second uses the values 200 for the weight, and 4 feet and 0 inches for the height

The expected outcome is a BMI of 62.5 and a return of Overweight

These are both general test cases to check that the math is correct and make sure the categories work properly. These are necessary to make sure the function works and outputs the correct data.

The third is to test the lowest possible outcome, with inputs of 0 for the weight and 1 foot, 0 inches for the height. This is to test the functions ability to handle the lowest outcomes.

The expected output is a BMI of 0 and a return of Underweight.

The last four test the boundaries of the categories, the fourth and fifth getting BMIs of 30.0 and 29.0, testing the boundary between Overweight and Average Weight. These are necessary to test the balance between the two boundaries.

The sixth and seventh get BMIs of 18.4 and 18.5, testing the boundary between Underweight and Average Weight. These are necessary to test the balance between the two boundaries.

The Retirement calculator has 5 test cases

The first three are general test cases for checking the math and categories. These are needed to ensure the function works as expected.

The last two test the boundary between the categories, with the fourth returning an age of 100 and a True for the 'Dead' variable, and the fifth returning an age of 99 and a False for the 'Dead' Variable. These are needed to test the balance between the two categories.

The testing is complete, since there are test cases for the possible inputs and boundary cases that can arise from use. For the BMI and Retirement Calculators, test outputs were high and low, and as close to the boundaries as possible.

I used a black box testing strategy by testing the functions as a whole and gauging their outputs from the inputs. To create test cases, I would first input my own data, being my height and weight, and used that as the first to measure the success of the function. I would then use numbers to intentionally get high or low outputs, or intentionally get close to the boundaries.

Execution Instructions

In order to execute these files, download them from the repository and put them all in the same folder.

Run all four of them to your machine.

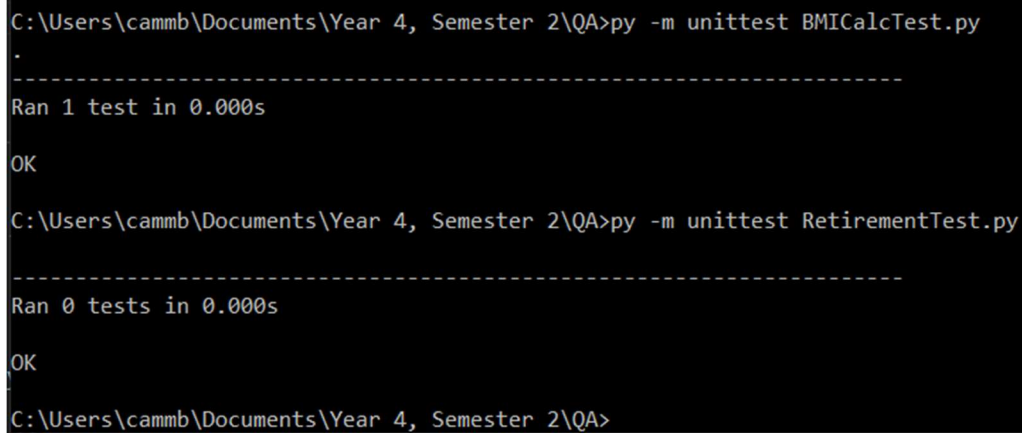
Open a terminal or command line in the folder with the files and type in

Py -m unittest BMICalcTest.py

And

Py -m unittest RetirementTest.py

This will give you the following output



```
C:\Users\cammb\Documents\Year 4, Semester 2\QA>py -m unittest BMICalcTest.py
.
-----
Ran 1 test in 0.000s

OK

C:\Users\cammb\Documents\Year 4, Semester 2\QA>py -m unittest RetirementTest.py
-----
Ran 0 tests in 0.000s

OK

C:\Users\cammb\Documents\Year 4, Semester 2\QA>
```

These commands run the test cases and reports the success or failure to the terminal.

It says 1 test but that is because each test case is in one function in the class, it does run each individually and check for success or failure. In the BMI test cases, if I change the third case expected output from 0.0 to 1.0, I will get the following output.

```
C:\Users\cammb\Documents\Year 4, Semester 2\QA>py -m unittest BMICalcTest.py
F
=====
FAIL: test_BMI (BMICalcTest.BMITest)
=====
Traceback (most recent call last):
  File "C:\Users\cammb\Documents\Year 4, Semester 2\QA\BMICalcTest.py", line 15, in test_BMI
    self.assertEqual(BMI,(1.0, 'Underweight'))
AssertionError: Tuples differ: (0.0, 'Underweight') != (1.0, 'Underweight')

First differing element 0:
0.0
1.0
- (0.0, 'Underweight')
? ^
+ (1.0, 'Underweight')
? ^

PRO
Ran 1 test in 0.001s

FAILED (failures=1)
C:\Users\cammb\Documents\Year 4, Semester 2\QA>
```

This shows that the tests are ran on each case and will catch errors.