Assignment 2

David Penfield, dep153

Github: dep153

BMI Calculator

BMI_Calc

BMI_Calc is a function meant to perform the math involved in calculating the BMI of a person when given their weight in pounds and their height in feet and inches. Tests for this function were built under the assumption of no user error. It assumes that any errors are on the side of the software. Because this function does not deal with boundaries, there are no edge cases to consider when testing. As a result, we only need to worry about the math in the function being correct and returning expected values for given input. As such, I included one test case for this function to do exactly that.

Get_Category

Get_Category is a function meant to return the BMI category of a person when given their BMI as calculated by BMI_Calc. Tests for this function were similarly built under the assumption of no human error, and that any error is on the part of the software. Unlike BMI_Calc, this function deals with several boundaries when comparing the given BMI value to the values that define each category. To deal with this, I utilized EPC to build test cases for this function that would handle BMI values within every possible category, as well as test cases to check the edge cases between these categories. The reason I did this was to ensure that any edge cases were handled correctly, and not somehow tossed into the bordering category by mistake.

Retirement Tracker

Retire_Tracker

Retire_Tracker is a function meant to perform the math involved in calculating how far a person is age-wise from their desired savings goal for retirement when given the person's age, annual salary, percentage of salary saved, and their retirement goal. Tests for this function were also built under the assumption of no user error, and that any error is the software's fault. Much like BMI_Calc, this function only deals with a simple math formula, meaning there are no boundaries to be tested. As such, this function was similarly tested with only one test case built to ensure that the function outputs the expected value for a given input.

The_Hard_Truth

The_Hard_Truth is a function meant simply to check whether a person will meet their savings goal within their lifetime given the age they are projected to meet their goal and returns the answer. Tests for this function were once again built under the assumption that any error is caused by faulty software rather than by user input error. An additional assumption is made, however, that 100 years of age is the projected age of death of the user. This function is like Get_Category, albeit simpler due to a smaller number of boundaries to deal with. As such, I apply the same testing strategy of EPC to test several cases just around the boundary point built into the function as well as both points well within their boundaries to ensure that all values return as expected. Because this covers all the cases capable of different errors, I have deemed it sufficient.

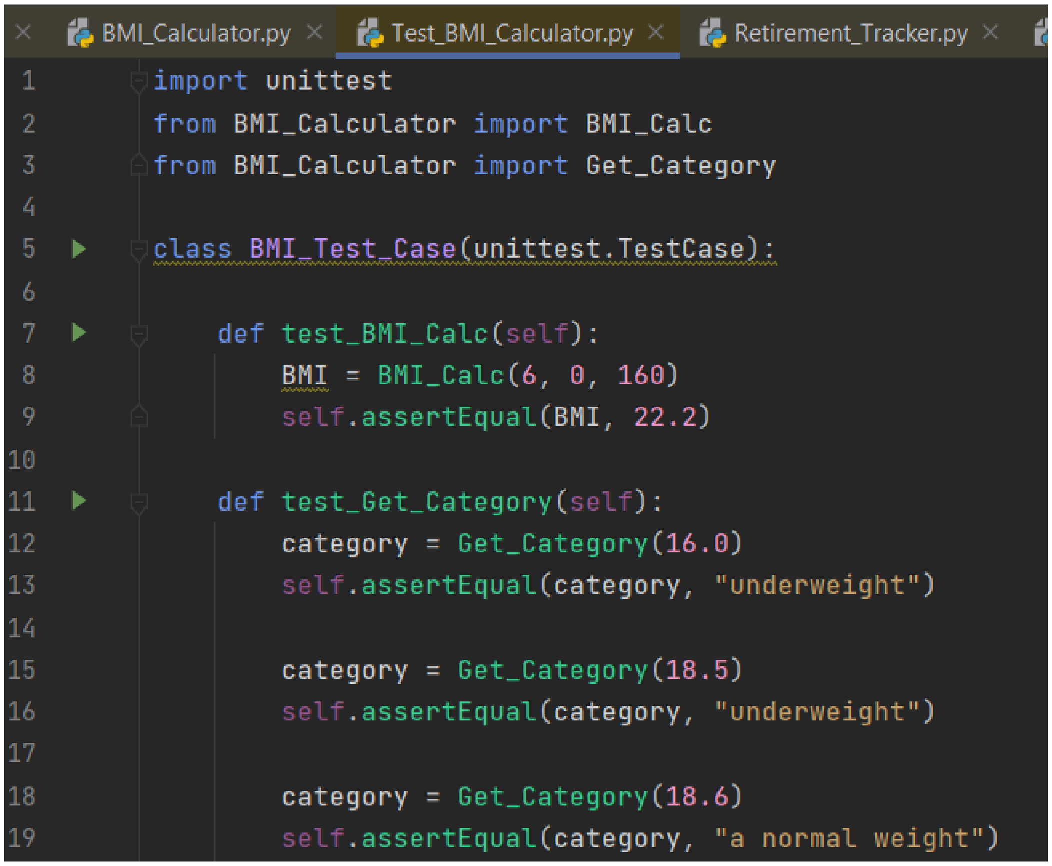
Setup

In building this project, I worked from a Windows 10 machine using Python version 3.9. The latest version of Python can be found here: https://www.python.org/downloads/. The IDE I used to put this together is called JetBrains Rider, which can be found here: https://www.jetbrains.com/rider/. This IDE is licensed, but students and teachers can obtain special permission to use it for free, and even if you are not a student or a teacher, you can still try it out for free for 30 days.

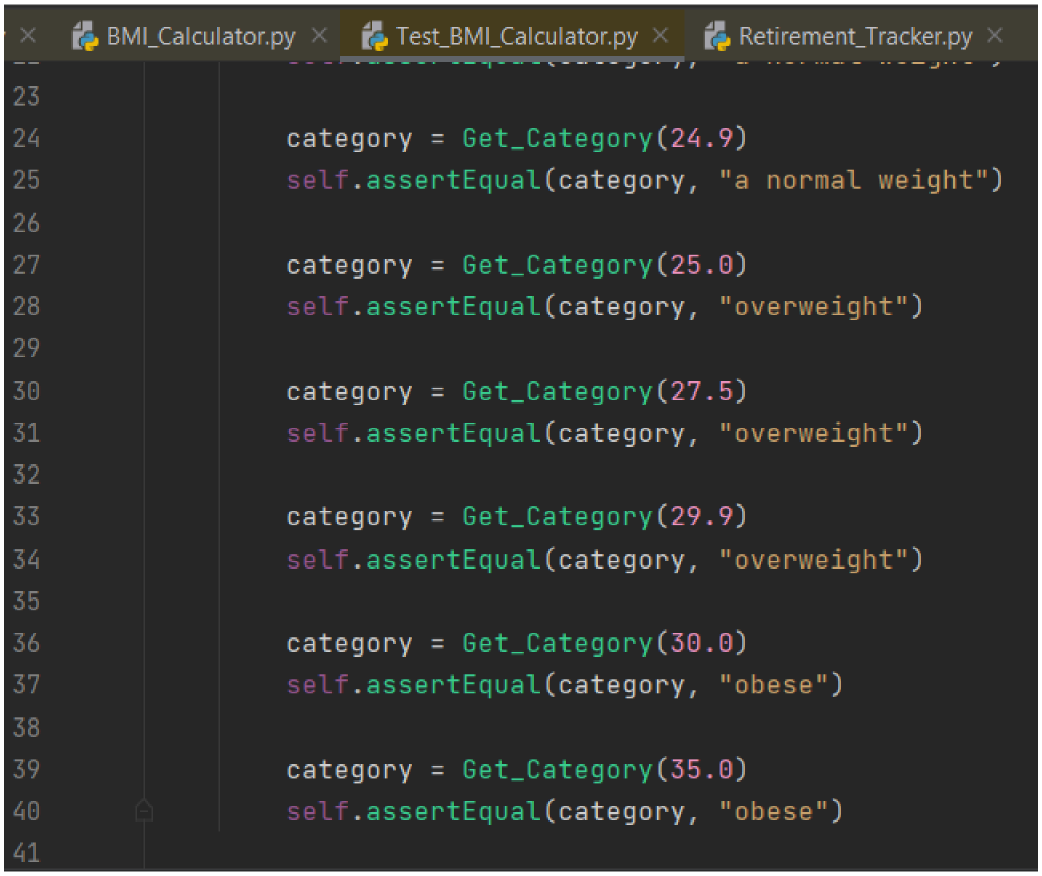
Execution

In order to be able to run this software you must first download the source code, which can be found on my personal github here: https://github.com/dep153/SoftwareQAAssignment2. Do this and choose where on your computer you wish to place the files. Before going any further, make sure that you have Python 3.9 downloaded and installed on your computer. Once you have downloaded the project, you can navigate to the directory you stored them in. Make sure that the files have not been separated at any point, and that all of them are in the same directory. Once you are sure of this, you can simply double-click the App.py file to launch the program. Success! Now you too can see what your BMI is, or discover how hopeless your plans for retirement are.

Screenshots

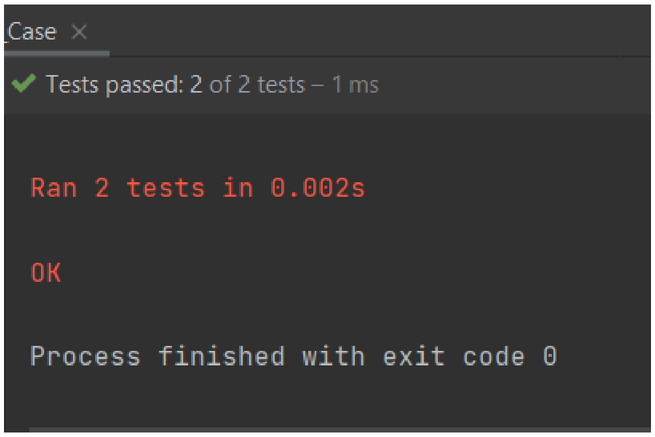


```python
import unittest
from BMI_Calculator import BMI_Calc
from BMI_Calculator import Get_Category

class BMI_Test_Case(unittest.TestCase):

    def test_BMI_Calc(self):
        BMI = BMI_Calc(6, 0, 160)
        self.assertEqual(BMI, 22.2)

    def test_Get_Category(self):
        category = Get_Category(16.0)
        self.assertEqual(category, "underweight")

        category = Get_Category(18.5)
        self.assertEqual(category, "underweight")

        category = Get_Category(18.6)
        self.assertEqual(category, "a normal weight")
```



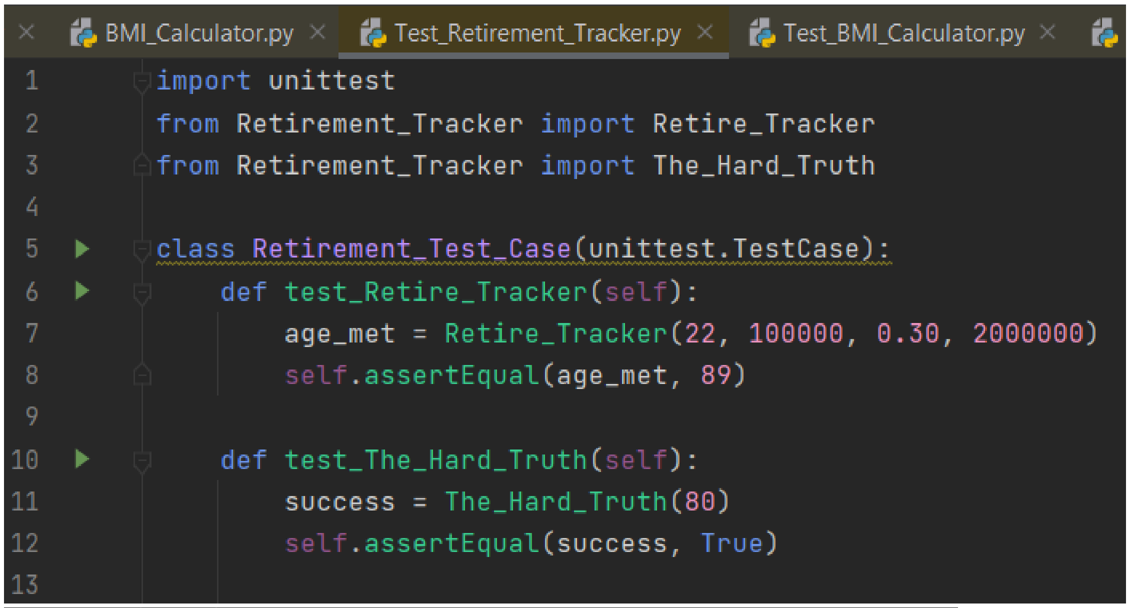
```
        category = Get_Category(24.9)
        self.assertEqual(category, "a normal weight")

        category = Get_Category(25.0)
        self.assertEqual(category, "overweight")

        category = Get_Category(27.5)
        self.assertEqual(category, "overweight")

        category = Get_Category(29.9)
        self.assertEqual(category, "overweight")

        category = Get_Category(30.0)
        self.assertEqual(category, "obese")

        category = Get_Category(35.0)
        self.assertEqual(category, "obese")
```

These are the test cases I built for my BMI Calculator. The first test case belongs to the BMI_Calc function and is testing to make sure the math is done correctly. The second test case belongs to the Get_Category function and is testing in and around every boundary to ensure the output returns as expected from in each case.

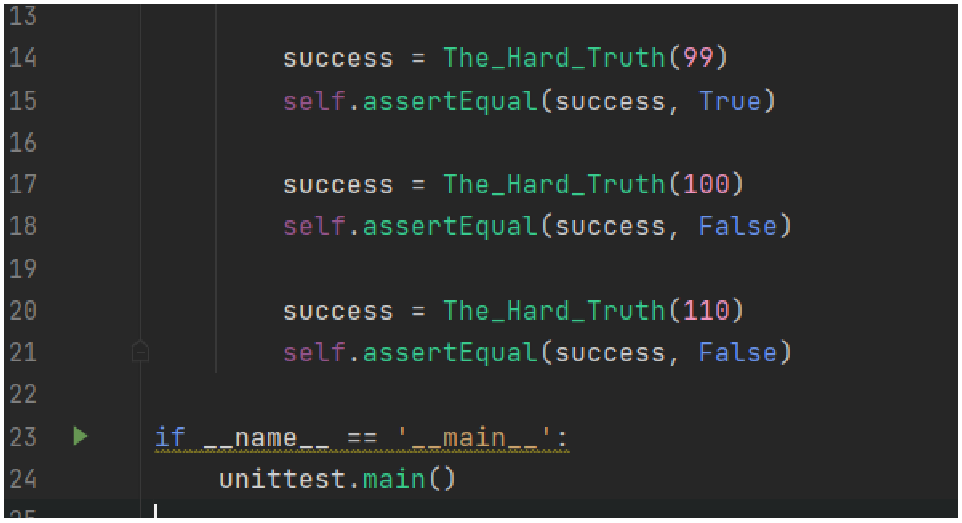


```
✓ Tests passed: 2 of 2 tests – 1 ms

Ran 2 tests in 0.002s

OK

Process finished with exit code 0
```

These tests thankfully pass without issue.

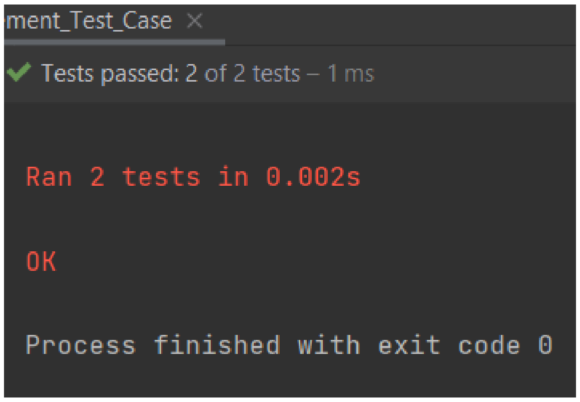


```python
import unittest
from Retirement_Tracker import Retire_Tracker
from Retirement_Tracker import The_Hard_Truth

class Retirement_Test_Case(unittest.TestCase):
    def test_Retire_Tracker(self):
        age_met = Retire_Tracker(22, 100000, 0.30, 2000000)
        self.assertEqual(age_met, 89)

    def test_The_Hard_Truth(self):
        success = The_Hard_Truth(80)
        self.assertEqual(success, True)

```



```python

        success = The_Hard_Truth(99)
        self.assertEqual(success, True)

        success = The_Hard_Truth(100)
        self.assertEqual(success, False)

        success = The_Hard_Truth(110)
        self.assertEqual(success, False)

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

These are the test cases I built for my Retirement Tracker. The first test case belongs to the Retire_Tracker function and is testing to ensure the math being done is correct. The second test case belongs to the function The_Hard_Truth and is testing each boundary case to make sure they return the expected values. These tests also passed without issue.



```
✓ Tests passed: 2 of 2 tests – 1 ms

Ran 2 tests in 0.002s

OK

Process finished with exit code 0
```