Week 6 Assignment - Testing and Refactoring

Overview

This week, we focused on how to build tests for our software and how to refactor our code to be simpler or more efficient. For this week's assignment, we are going to build some functions that will help us do some temperature unit conversions. We will start by writing out tests that fail, and then building some code to make that test pass, until we have a working version of the application. After that, we will see how we can potentially refactor this implementation into something simpler.

Our code will be organized into two files:

1. *conversions.py*, which holds functions that will do these temperature unit conversions; and
2. *tests.py*, which will hold all of our tests

Make sure to create a github repository for this assignment named **IS211_Assignment6**. All development should be done in this repository.

Useful Reminders

1. Read the assignment over a few times. At least twice. It always helps to have a clear picture of the overall assignment when understanding how to build a solution.
2. Think about the problem for a while, and even try writing or drawing a solution using pencil and paper or a whiteboard.

Part I - Create Tests for Celsius Conversion

Following the *test-driven development* methodology, we are going to start by first creating some tests for our implementation. Let us start with testing two functions that we will eventually need to write and put inside *conversions.py*:

1. `convertCelsiusToKelvin` - Takes in a float representing a Celsius measurement, and returns that temperature converted into Kelvins
2. `convertCelsiusToFahrenheit` - Takes in a float representing a Celsius measurement, and returns that temperature converted into Fahrenheit

For now, create these functions in the correct file but have them return 0.0.

The tests should make multiple calls to each function, checking if the conversion is correct. For example, 300.00 °C is 572.00 °F and 573.15 K. For more information on how to do these conversions, please refer to Conversion of Units of Temperature webpage, which has a reference table of known conversions that you can use for your testing. Make sure this test is thorough by including 5 test cases per test. Also make sure to be verbose in your testing (i.e. print a message for every test case you are checking).

At this point, since those functions have not been written properly, the tests will fail. Let us correct this in the next section.

Part II - Implement the Celsius Conversion Functions

Now we need to make the tests work. Do this by correctly implementing the conversion functions listed above. Make sure all your tests at this point run without any errors.

Part III - Repeat

Repeat Part I and II for converting from Fahrenheit to Celsius and Kelvin, as well as converting from Kelvin to Fahrenheit and Celsius. There should now be a total of 6 functions, each one having a test case which confirms it is properly converting. Make sure all your tests can run successfully.

Part IV - Refactor

In many real world situations, the requirements for your code may change. It behooves you as a programmer to think in general terms, such that if requirements do change, you can minimize the impact of how your code needs to change. Lets mimic this by changing the requirements of the application to support not only temperature, but to support converting between Miles, Yards and Meters.

Now, to support this, we can do like we did above: keep creating functions to convert one unit to another, and writing a test for that function to confirm its functionality. However, is there a more general way to solve these conversion problems? Do they have a similar pattern? If they do, is there a way for you to refactor your code such that you only need one functions to convert any unit into another (of course, we can't convert Celsius to Meters, and we should account for that by throwing an exception).

Lets create a new method inside of a new file (called *conversions_refactored.py*) named `convert` that takes in 3 values:

1. fromUnit - the unit we are converting from, as a string
2. toUnit - the unit we are converting to, as a string
3. value - the value of fromUnit we are converting from

The function should return the converted value as a float. This method should not rely on the methods we wrote above. This about the problem and see if a pattern emerges that lets you implement it in a simpler way. The test for this function should:

1. Check that all temperature conversions are working
2. Check that all distance conversions are working
3. Check that converting from one unit to itself returns the same value for all units
4. Check that converting from incompatible units will raise a `ConversionNotPossible` exception (which should be defined in the *conversions_refactored.py* file)