HW 5: For loops and sequences (lists / tuples)

Due October 20th, 6pm, codePost.io
Please submit all files created for this assignment to codePost. You can submit as many times as you like up until the deadline. ***If you are planning to use one or more late days, please notify us by sending a private Piazza message to "Instructors".***

Familiarize yourself with how to approach a programming assignment and review the grading rubric on this assignment's page in Canvas before getting started.

Code style, documentation, and other requirements
All code style, documentation, and other requirements from the previous assignments apply to this one, too, with the exception of `main()`. You do not need to write a `main()` function for any of the problems in this assignment.

We will be checking that you are using GitHub regularly (at least 4 commits per homework). GitHub usage counts toward your professional skills score.

There is no written component for this assignment.

Programming Component (100 pts)

Problem 1: upc.py and test_upc.py

A Universal Product Code (UPC) is a special number that you will find near the barcode on most products you buy. Most of the digits in the UPC specify product information such as the manufacturer, country of origin, etc., but one of the digits is set aside to be the "check digit". After all the other numbers are set, the check digit is chosen such that a calculation we apply to the numbers will end being a multiple of 10. UPCs are designed this way to make it easy to tell if a UPC has been entered incorrectly during checkout.

Write a function called `is_valid_upc` that takes a string as input and returns True if the supplied string is a valid UPC or False if it's not.

The algorithm to determine whether a number is a valid UPC is as follows. The validation algorithm proceeds on the number **from right to left**:

- Leave digits in odd even positions, including zero, as they are.
- Multiply digits in odd positions by 3.

- Sum the results. If the total is a multiple of 10, the supplied number is a valid UPC, otherwise it's invalid.

An example
The UPC in the photo above is 9 7 8 0 1 2 8 0 5 3 9 0 4. We apply the algorithm (remember it goes from right to left so 4 is at position 0, 0 is at position 1, etc:

```
      4 + 0 * 3 + 9 +  3 * 3 + 5 + 0 * 3 + 8 + 2 * 3 + 1 + 0 * 3 + 8 + 7 * 3 + 9
=     4 + 0 + 9 + 9 + 5 + 0 + 8 + 6 + 1 + 0 + 8 + 21 + 9
=     80
```

80 is a multiple of 10 so this is a valid UPC.

Caution:
- UPCs can be different lengths.
- Additionally, don't assume the supplied string contains only digits. You are not expected to try to convert the input to the correct format (e.g. by removing spaces from a string that otherwise contains only digits). If the supplied string contains characters other than digits, the function should return False because the string is not a valid UPC.

Don't forget to test your function thoroughly in test_upc.py!

Problem 2: password.py and test_password.py

In password.py, write a function called `secure_password` that takes a string and checks if it is a secure password according to the following rules:
- It is between 9 and 12 characters long (inclusive)
- AND it meets at least 3 of these requirements:
    - At least one lowercase letter
    - At least one uppercase letter
    - At least one digit (0-9)
    - At least 1 of the following special characters: $, #, @, !
- AND it does not contain any special characters other than $, #, @, or !

Hint: Boolean values can be treated as integers. True is equivalent to 1 and False is equivalent to 0. In the Python interpreter, try performing calculations with boolean values e.g. True + True. This may be useful for this problem!

You are not required to write main() as we'll be using our own Pytest tests on your function.

Don't forget to test your function thoroughly in test_password.py!

Problem 3: lightrail.py and test_lightrail.py

For this problem, you're practicing test-driven development (light version). This means we're providing you with the function docstrings and Pytest test functions. Download the starter files from the lecture-code repo > Starter code > HW 5. Your task is to write the rest of the program, including defining the functions described in the docstrings in lightrail.py, which are called by the tests in test_lightrail.py.

You're writing a module containing functions for getting basic directions on the Seattle Link light rail. The starter file includes a predefined tuple constant that contains all valid station names in order from north (University of Washington) to south (Angle Lake). You can use this information to work out the direction of travel, "Northbound" or "Southbound".

You may add additional helper functions (you will probably need at least one) to lightrail.py. If you do, make sure to document it and test it in test_lightrail.py. Do not change any of the existing tests.