**4PINT - Module 5 Labs**

In each lab you'll work through a series of exercises that help you to understand and be able to apply and use the knowledge and techniques learnt in the associated weeks' lecture. Remember – you aren't expected to do everything from memory – sometimes it's best to do a little research / reading / rereading of materials and come up with a ***great***, and **correct**, answer rather than just 'taking a stab' at a question or exercise and hoping you're right!

## Lab 01

Write a Python function that takes the marks as a parameter and returns the equivalent grade based on the following rules:

| Marks | Grade |
|---|---|
| 85-100 | HD |
| 84-75 | D |
| 74-65 | C |
| 64-50 | P |
| 49-0 | F |
| All other values | ERROR |

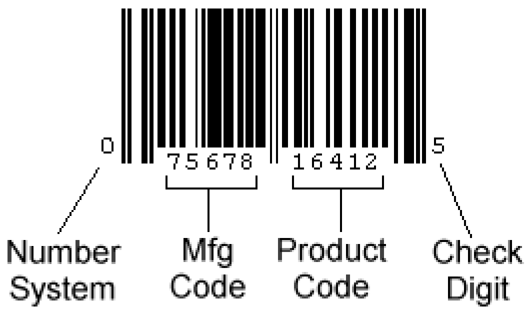
Test this function by creating a suitable test class that uses Pythons **unittest** module. Test for all values as above

## Lab 02

Open the **Module 3 Lab Exercise** file and using the **Stock** class you created in **Activity 02**,write a test class using **unittest** to test the functionality of this class

## Lab 03

A **check digit** is a single digit in a larger value that is used to determine whether the value as a whole has not been modified or corrupted and is valid. Lots of things have check digits embedded in them to ensure the number provided is legal - for example bar codes on products, credit card numbers, tracking numbers on parcels etc.



You are provided with a class called **UniversalProductCode** (available in the Module 7 Resources folder) which calculates the check digit of a value using the follow process:

a) Add the digits in the odd-numbered positions (first, third, fifth, etc.) together and multiply by three.
b) Add the digits (up to but not including the check digit) in the even-numbered positions (second, fourth, sixth, etc.) to the result.
c) Take the remainder of the result divided by 10 (modulo operation) and if not 0, subtract this from 10 to derive the check digit

Now that we have a class to test, create a **upc_test.py** file in which you should write a **TestUniversalProductCode** class that uses Python's **unittest** module to test that various product codes are valid or invalid, and that providing data in the wrong format (i.e. a product code that contains letters is not valid).

You can look up some valid universal product codes (for example, here's some for Cadbury chocolate bars – but you can search for anything: http://www.upcitemdb.com/info-cadbury_bars – add a zero at the beginning if valid codes looked up say they are not valid!). However, to save you some time here are a few to work with:

|  |  |
|---|---|
| - 012345678905 | ← Valid |
| - 012345678906 | ← Invalid check digit |
| - OneTwoThree | ← Invalid input |

In your **TestUniversalProductCode** class you should write a test that checks for a few different valid and invalid UPC codes, and modify the UniversalProductCode class to raise an exception instead of crashing when provided with illegal data (i.e. 'OneTwoThree' is illegal because it doesn't contain only digits).

Use the unittest **setUp(self)** method to provide a number of UPC codes in a list (see slide 31 for details on how to use the setUp method, and slides 22 onwards for details on exception handling).

For the above three UPCs, you should use the **assertTrue**, **assertFalse** and **assertRaises** assertions to ensure the first (valid) code is accepted as valid, the second isn't, and the third throws an exception.