Assignment 1: Basic Training and Lists

Due Tuesday 24/3 @ Midnight

Submit using the online submission system.

Assignment aim and objectives

The aim here is to implement correctly four functions:

- distance(t, accel): calculate distance travelled in t seconds at accel m per seconds squared
- pythagoras(a, b): calculate the length of the hypotenuse of a right angled triangle
- grade(mark, mcr): calculate a letter grade based upon a mark and whether the student has met the mandatory course requirements
- print_before(text,marker): take a list of words, print out (but not including) the first occurrence of a marker word

Correctness is determined by running against regression tests and by the tutors reading your code.

The objectives of the assignment are to: (1) write correct code applying what was learnt in lectures; (2) gain experience at using a unit testing framework for Python (doctest this week); and, (3) write readable Python code that follows commonly used stylistic rules.

You can obtain all the files you need from this gzipped tar archive:

- assignment1.tar.gz: assignment1.tar.gz

- assignment1.zip: assignment1.zip

Where to put your code and guidance on style

All of these functions are defined in **`assignment1.py`**. You will find the name of the function, a header defining arguments and documentation of what it should be doing. For example:

```
def distance(t, accel):
    """Print the distance travelled.

    Calculates the distance travelling in a
    given time (t) at a given acceleration (accel).

    You should round the result to ONE decimal
    places (use the round method, for example
    round(10.01,1) creates the output 10.0).

    Why do this? To hide any problems related to
    floating point precision that would make automated
    testing harder.

    Args:
       t is an integer
       accel is a float

    """
    print(t,accel)
```

Line 01 defines the function **`distance(...)`** in terms of its name and its formal arguments (note unlike Java **no typing** of formal arguments, pass what you like and hope the function works!).

Lines 02-20 are the body of the function. Note also that there are no use of braces to define the scope of the function, instead indenting and a final blank line (not shown, follows line 20) play this role.

Lines 02-19 are documentation, each of the functions has documentation setting out the requirements for the completed function. You must implement these.

At the moment the implementation (line 20) simply prints out to the console the actual arguments passed to the function.

You can run this (use Python 3.x) by following these steps (note *do not type $ or >, they just represent the prompts for the Unix shell and the python interpreter!):

```
$python3
>from assignment1 import *
>distance(10,100)
```

Line 01 invokes the Python interpreter. Line 02 imports the code defining the functions. Line 03 executes the function with two arguments.

Your job is to complete the implementation of each function using what we have covered so far in "Basic training" and "Lists". You don't need to use the `return` statement anywhere, all output should simply be printed and in fact **you are not allowed to use break or return anywhere in your code**.

When writing your code you should follow some fairly standard stylistic rules for Python.

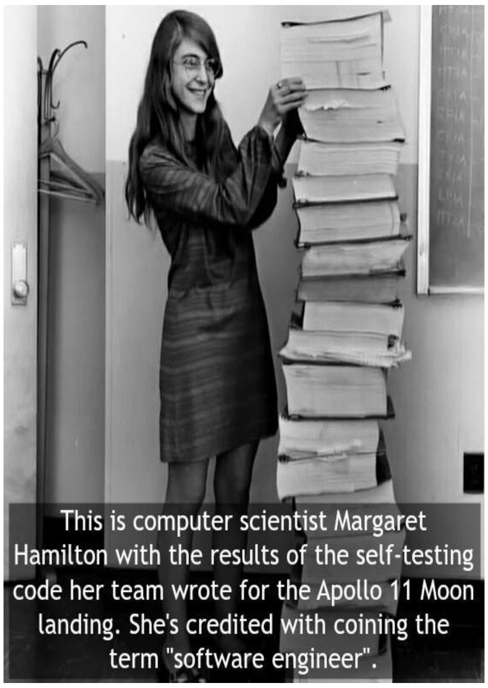
Naming conventions Follow these formats: `function_name`, `function_parameter_name` and `local_var_name`.

Use comments to document anything non-obvious You can use the # character for single line comments or use multiline comments that are bracketed by """ (look at the multiple line comments for the functions as a guide).

Don't put commands on same line separated by semicolons I do this in the workbooks for brevity but its considered bad for programs that are more than a line or two long.

Testing your code (plus extra requirements for correctness)

Testing is really important so both Python assignments and project will require testing. The idea of self-testing code has been around a while (see below) and we apply this idea in this project.


This is computer scientist Margaret Hamilton with the results of the self-testing code her team wrote for the Apollo 11 Moon landing. She's credited with coining the term "software engineer".

We are using a testing framework called doctest. It is a simple framework that suffices for this assignment.

The files `a1_xxxx_test.txt` (where `xxxx` refers to the function being tested) defines some simple unit tests for regression testing of the code. Below is the section defining the tests for `distance`:

```
This is a doctest based regression suite for assignment1.py
Each '>>' line is run as if in a python shell, and counts as a test.
```

```
The next line, if not '>>' is the expected output of the previous line.
If anything doesn't match exactly (including trailing spaces), the test
fails.


Import the functions
>>> from assignment1 import *

Test the distance calculation
>>> for t in range(0,10):
...     distance(t,9.8)
...
0.0
4.9
19.6
44.1
78.4
122.5
176.4
240.1
313.6
396.9
```

Each function has its own set of tests. The aim is to have no failures when you run automatic testing using this command:

```
python3 -m doctest -v a1_distance_test.txt
```

```
Trying:
    from assignment1 import distance
Expecting nothing
ok
Trying:
    for t in range(0,10):
        distance(t,9.8)
Expecting:
    0.0
    4.9
    19.6
    44.1
    78.4
    122.5
    176.4
    240.1
    313.6
    396.9
ok
1 items passed all tests:
   2 tests in a1_distance_test.txt
2 tests in 1 items.
2 passed and 0 failed.
Test passed.
```

The good news here is on line 23 that says both the tests passed and nothing fails. When you do have errors the output will tell you what the test was, what was expected as output and what was generated. Note that the **`import`** command that loads the function is counted as a test and you pass this as long as Python doesn't find syntax errors in your code so normally you will pass at least one test!

Treat the tests as an additional set of specifications for your function. Note that you can't just hardcode the results, that would not result in passing this assignment!

Completion and marking

The basic idea is to implement each function and test it against the regression tests. Submit your code once you are satisfied that you have correctly implemented the functions.

Do not change the file names or function names. Our automated testing will fail and you might not get any marks.

The majority of your grade (90%) is based upon correct implementation as evidenced by passing the regression tests that we have provided and passing our own secret regression tests. The remaining minority of your grade (10%) is based upon adherence to the style guide and general readability of the code (i.e. don't over complicate it!).

Model solutions to the assignments will be released shortly after the assignment deadline (printed on paper). This means late submissions will NOT be accepted, unless you have made prior arrangement with the course coordinator for valid reasons such as medical and family emergencies.