# U08008 Coursework 1

In this coursework you will be assessed on learning outcomes 1, 2 and 3 as given below.

*On successful completion of this module, students will be able to:*

|  |
| --- |
| 1. *Design, implement and test computer programs, derived from application based case studies, implemented using a modern computer programming language.* |
| 1. *Demonstrate a thorough understanding of the fundamental concepts of high level programming languages, including the syntax and semantics of constructs for input/output, control flow and elementary data structuring.* |
| 1. *Appreciate the steps involved in translating a program description expressed in a high level language to a run time executable form.* |

You are required to provide an informal design, implementation and informal testing on a program to work out a student’s average mark for the modules they have taken so far and how well they will need to do on subsequent modules to achieve a specific grade. Brookes operates both the Grade Point Average and traditional British classification systems. This program will implement a simplified version of the Brookes rules for classification using the traditional British system.

## Rules for Classification (simplified)

1. First year modules are not part of the classification and will not be considered for this assignment.
2. 16 modules must be passed in the second and final year
3. Of these only the top 15[[1]](#footnote-1) passes are included in the calculation
   1. The lowest mark is discarded
   2. The final average is then worked out by dividing the total of all the marks by 15
4. The classification uses a simple average of these 15 scores to decide the grade as shown in the table below.

|  |  |  |
| --- | --- | --- |
| Average | Classification | Also Known as |
| >=70% | First Class Honours | 1 |
| >=60% | Upper Second Class Honours | 2:1 |
| >=50% | Lower Second Class Honours | 2:2 |
| >=40% | Third Class Honours | 3 |

## Your Program

Your program should first allow the user to enter all the marks that they have gained so far in the second and third year. The program must ask for the marks without taking into account the module name, the year taken or the type of module. You can assume that all the modules are single modules.

The program should be keeping track of

* the total number of module marks entered so far,
* how many marks are passes,
* how many marks are fails,
* the total of all the pass marks.

To do this it should go into a loop that will keep asking the user for another mark until they say they do not wish to enter any more, or 16 pass marks have been entered. Your program should include data validation on the marks entered. All marks should be between 0 and 100 inclusive. If the user attempts to enter a mark that is less than 0 or greater than 100, the program should go into a loop that keeps telling them when they are wrong and asking them to enter a valid value until they do so.

Once the marks have been entered; the program should calculate and display the current average mark across all the passes, even the lowest one. It should then tell the user what their current average is and ask them what they are aiming for. The program should then display the grade and average information that is shown in the table above to remind the user and then store the desired average mark entered by the user.

The program will then work out what average mark the user will need over the remaining modules to reach the overall average mark and report this to the user.

## Working out the required average

The degree classification is a simple average over the 15 highest pass marks. When an average mark is known, the total number of marks that it represents can be worked out by simply multiplying the average by 15.

When the user specifies an average they are aiming for, it is thus possible to work out how many marks are required to reach that average. From the current average, it is possible to work out the total marks the student has so far and by subtracting that total from the total required for the desired average; the number of marks that are still needed can be worked out. If this number of marks still needed is divided by the number of modules left, a required average can be calculated and displayed to the user.

As the lowest mark is discarded, once the total marks for the modules currently passed is calculated, the lowest mark should be subtracted from it and the module count reduced by 1.

e.g.

*11 modules passed with an average of 56% the total mark is 56 x 11 = 611 marks*

*If the lowest mark is 51% then this is removed and the count reduced by 1 to give*

*10 modules passed with a total of 611 – 51 = 560 marks.[[2]](#footnote-2)*

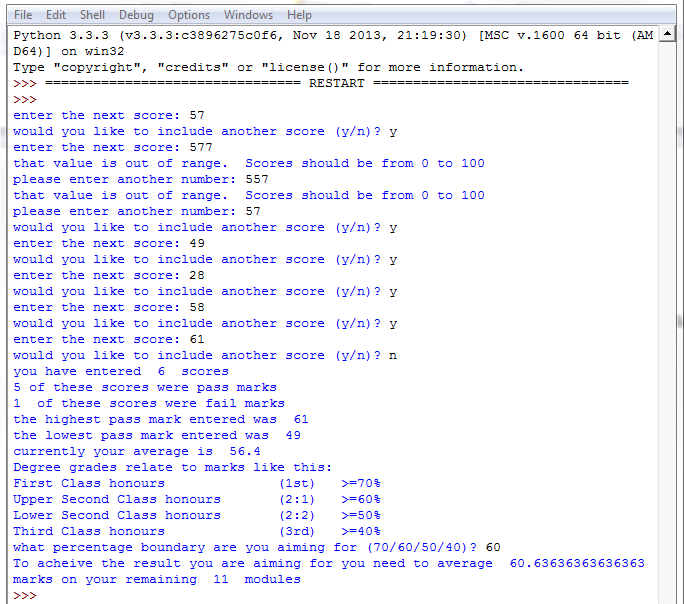
*These marks count towards the classification*

*If the user wants a final average of 60% the total is at least 15 x 60 = 900 marks*

*The number of marks still required to get that average is 900 – 560 = 340 more marks.*

*Since the user has 5 modules left to score 340 marks they need to average at least 340/5 = 68%[[3]](#footnote-3)*

The screenshot below shows a program that meets the specification in operation.



## What you need to do

Have a look at the example assignment to get some idea of how you should lay things out and what is required for each section.

You should first design the program informally. The design you submit should take the form of an explanation of why your program is structured the way it is and how the code works. It should show what data will need to be used by the program and detail the steps to use it.

You should next create a well structured python programme that uses a lot of the concepts that we have learned up to this point and makes sensible use of comments.

NOTE: this assignment is testing how well you can solve problems with the limited set of tools we have introduced so far. It is not a test of how well you know the python libraries. So for example, you should write the code to calculate an average yourself rather than looking up a python average function.

You should carry out informal testing of your program. This will involve checking that all the calculations work and testing that all parts of the code do what they need to. For example does the program ask the user to enter correct values when they enter out of range ones? When testing the calculations you should first carry out the algorithm by hand to see what values it should produce then enter the same values into the computer.

## Assessment

This assignment is assessed using the following. In each case it shows what will usually be the case for each of the criteria:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pass | Good Pass | Excellent Pass |
| Informal Design  5 marks | Shows some understanding of how to solve the problem  Explains how the code will work. | Shows a good understanding of how to solve the problem  Explains clearly how the code will work with a logical structure | Solves the problem correctly and efficiently. Clear explanation of how the code will work with a logical structure and obvious correspondence between design and code |
| Code: use of Python  3 marks | Shows some familiarity with how to use the various python commands and produces a working program. | Shows familiarity with how to use the various python commands correctly and uses them in a sensible way throughout the code | Shows a deep understanding of how to use the python constructs and as a result produces efficient code |
| Code: Problem Solving  5 marks | Solves most of the problem may struggle with working out the average required on future modules | Solves the problem correctly. | Produces neat efficient code that solves the problem in a highly competent manner. |
| Code: Complexity  5 marks | Makes a reasonable attempt to carry out the calculations sensibly | Carries out all the calculations and shows an understanding of how to properly implement them in python | Program does all that is asked of it |
| Code: Structure  2 marks | Code has a logical progression but may meander about to get to the answer. | Code is logical and systematic throughout. Structure of code suits the problem. | Neat efficient code that is well structured and precise. |
| Informal Testing  5 marks | Calculations done by hand are correct and match what the computer produces | In addition, the data chosen ensures that the different types of response by the program are checked. | Testing is accurate and comprehensive. |

## Submission

You should submit the following:

1. Your informal design of your program. **[5 marks]**
2. A .py file containing your python code **[15 marks]**
3. Your informal testing **[5 marks]**

The work you submit should comprise of two files. A Word file (or equivalent that can be read from MS Word in the pooled rooms) and a .py file. The word file should contain the design, testing and the code copied and pasted in it. The .py file should be a working python file containing your code. The work should be submitted through Moodle.

1. The top 14 are actually used in the real Brookes calculation but it is far easier to only keep track of one lowest score using what we know of python so far. There are also limits on which modules can be discarded which are being ignored for simplicity. [↑](#footnote-ref-1)
2. This assumes all future marks will be higher than the current lowest. Not necessarily true but used for simplicity. [↑](#footnote-ref-2)
3. There are also near miss rules but again, these are being ignored for simplicity [↑](#footnote-ref-3)