Programming Principles (CSI6208)

Assignment 1:	Individual programming assignment ("Number List Test" program)
Assignment Marks:	Marked out of 30, worth 30% of unit
Due Date:	Check Canvas

Background Information

This assignment tests your understanding of and ability to apply the programming concepts we have covered in the unit so far, including the usage of variables, input and output, data types and structures, selection, iteration and functions. Above all else, it tests *your ability to design and then implement a solution to a problem* using these concepts.

Academic Integrity and Misconduct

The **entirety of your assignment must be your own work** (unless otherwise referenced) and produced for the current instance of the unit. Any use of Generative AI e.g., ChatGPT and unreferenced content you did not create constitutes plagiarism, and is deemed an act of academic misconduct. All assignments will be submitted to plagiarism checking software which includes previous copies of the assignment, and the work submitted by all other students in the unit.

Remember that this is an **individual** assignment. *Never give anyone any part of your assignment – even after the due date or after results have been released. Do not work together with other students on individual assignments – you can help someone by explaining a concept or directing them to the relevant resources, but doing any part of the assignment for them or alongside them, or showing them your work, is inappropriate.* An unacceptable level of cooperation between students on an assignment is collusion, and is deemed an act of academic misconduct. If you are uncertain about plagiarism, collusion or referencing, simply contact your tutor, lecturer or unit coordinator.

You may be asked to **explain and demonstrate your understanding** of the work you have submitted. Your submission should accurately reflect your understanding and ability to apply the unit content.

Assignment Overview

In this assignment you are required to implement a "Number List Test" program that generates lists of random integers and asks the user questions about them. At the start of the program, the user can select the difficulty of the test, which determines the number of questions in the test, as well as the quantity and range of the numbers. At the end of the test, the user's score is displayed.

The entirety of this program can be implemented in under 140 lines of code (although implementing optional additions may result in a longer program). *This number is <u>not</u> a limit or a goal – it is based upon an average quality implementation that includes comments and blank lines.*

Tip: You can (and should) make a start on this assignment as early as the end of Week 2.

When starting out with programming, it is common to encounter difficulties that temporarily slow or stop your progress. The sooner you start, the more time you have to work through difficulties.

Program Output Example

To help you visualise the program, here is an example screenshot of the program being run:

```
Welcome to the Number List Test program.
Select a difficulty:
[E]asy
[M]edium
[H]ard
> medium
Invalid choice!  Enter E, M or H.
> M
Medium difficulty selected!

Question 1 of 4.
What is the average of the numbers in this list? [5, 12, 8, 4, 6]
(round UP to nearest integer)
> 8
Incorrect!  Correct answer was 7.

Question 2 of 4.
What is the sum of the numbers in this list? [3, 7, 4, 4, 8]
> 26
Correct!

Question 3 of 4.
What is the average of the numbers in this list? [6, 11, 12, 11, 7]
(round UP to nearest integer)
> 10
Correct!

Question 4 of 4.
Challenge question!
What is the biggest number in this list? [23, 9, 24, 21, 17]
> 24
Correct!

Test complete!
You scored 3/4 (75.0%).
```

The program welcomes the user and then prompts them to select a difficulty. The user first tried entering “medium” and was told it was an invalid choice. They enter “M” and the program confirms that they have selected the medium difficulty. *The difficulty determines the number of questions in the test, how many random numbers are in each list of numbers, and range of those numbers (i.e. the minimum and maximum possible numbers).*

The program then asked the user 4 randomly selected questions – the type of question is randomly selected each time, allowing the same type of question to be asked multiple times. As you can see, a new list of random numbers is generated for each question. The final question is a “challenge question”, using a range of numbers that is double the normal range for the selected difficulty.

The user answered the first question incorrectly, but answered the remaining questions correctly. The program tells the user whether their answer is correct or not before proceeding to the next question, and then displays their score after the final question.

Tip: While working on your code, you may want to make the program print the correct answer to each question after asking it. This will save you time and be very useful for testing and debugging.

Pseudocode

As emphasised by the case study of Module 5, it is important to take the time to properly *design* a solution before starting to write code. Hence, this assignment requires you to *write and submit pseudocode of your program design* as well as the code for the program. Furthermore, while your tutors are happy to provide help and feedback on your assignment work throughout the semester, they will expect you to be able to show your pseudocode and explain the design of your code.

You will gain a lot more benefit from pseudocode if you actually attempt it *before* trying to code your program – even if you just start with a rough draft to establish the overall program structure, and then revise and refine it as you work on the code. This back and forth cycle of designing and coding is completely normal and expected, particularly when you are new to programming. *The requirements detailed on the following pages should give you a good idea of the structure of the program, allowing you to make a start on designing your solution in pseudocode.*

See Reading 3.3 for tips and advice regarding pseudocode.

Write a *separate section of pseudocode for each function* you define in your program so that the pseudocode for the main part of your program is not cluttered with function definitions. Ensure that the pseudocode for each of your functions clearly describes any parameters that the function receives and what the function returns back to the program.

Your solution must be modular and therefore more than one function is required in this assignment (detailed later in the assignment brief).

Tip: Do not attempt to implement the entire program at once. Work on one small part (a single requirement or even just *part* of a requirement) and only continue to another part once you have made sure that the previous part is working as intended *and that you understand it*.

It can also be useful to create separate little programs (e.g., functions) to work on or test small sections of code. This allows you to focus on just that part without the rest of the program getting in the way, and allows you to work on a section without necessarily having completed prior/other sections.

Program Requirements (Implement these requirements in "main.py" file)

In the following information, numbered points describe a *core requirement* of the program, and bullet points (in italics) are *additional details, notes and hints* regarding the requirement. Ask your tutor/unit coordinator if you do not understand the requirements or would like further information.

1. Print a welcome message, then prompt the user to select a difficulty by entering "E", "M" or "H". Use a loop to re-prompt the user until a valid choice is entered. Once a valid choice has been entered, print a message confirming the selected difficulty, and set variables as follows:

- *Easy difficulty… `questions` = 2, `quantity` = 3, `minimum` = 1, `maximum` = 5.*
- *Medium difficulty… `questions` = 4, `quantity` = 5, `minimum` = 3, `maximum` = 12.*
- *Hard difficulty… `questions` = 6, `quantity` = 8, `minimum` = 10, `maximum` = 25.*

- *"`questions`" represents how many questions the test contains, "`quantity`" represents how many numbers are in each list, and "`minimum`" and "`maximum`" represent the smallest and biggest number that can be randomly selected in each list. e.g. In easy difficulty, the test consists of 2 questions about lists of 3 random numbers between 1 and 5.*

- *The program should show an "invalid choice" message if the user does not enter "E", "M" or "H".*
- *The program should also allow "e", "m" or "h" – i.e. it should not be case-sensitive.*

2. Set a "`score`" variable to 0, and then enter a loop that repeats `questions` times.

- *"`score`" will be used to keep track of how many questions the user has answered correctly.*

The body of this loop must…

2.1. Print which question the user is up to out of the total number of `questions`, e.g.

```
Question 1 of 4.
```

2.2. If the user is *not* up to the final question, use the "`random_list()`" function (detailed below) to generate a list of `quantity` random numbers between `minimum` and `maximum`.

Otherwise (if the user *is* up to the final question), print "Challenge question!" and use the "`random_list()`" function to generate a list of `quantity` random numbers between *double* the `minimum` and *double* the `maximum`.

- *e.g. The challenge question of a medium test would involve a list of 5 numbers between 6 and 24.*

2.3. Generate a random number between 1 and 4 and use it to choose which question is asked.

- *Use the "`random.randint()`" function to generate a random number. Note that this is not the only approach you can take to selecting a random question. You may take a different approach as long as the requirements are implemented – consult with your tutor if in doubt!*

2.4. If the random number is 1, ask the user "What is the smallest number in this list?" and display the list of numbers generated in Requirement 2.2. If they answer correctly, print "Correct!" and add 1 to their `score`. Otherwise print "Incorrect!" as well as the correct answer.

- *The "`min()`" function can be used to determine the correct answer.*
- *Convert the user's input to an integer so that you can properly compare it to the correct answer.*

2.5. Otherwise, if the random number is 2, ask the user "What is the biggest number in this list?" and display the list of numbers generated in Requirement 2.2. If they answer correctly, print "Correct!" and add 1 to their `score`. Otherwise print "Incorrect!" and the correct answer.

- *The "`max()`" function can be used to determine the correct answer.*
- *Convert the user's input to an integer so that you can properly compare it to the correct answer.*

2.6. Otherwise, if the random number is 3, ask the user "What is the sum of the numbers in this list?" and display the list of numbers generated in Requirement 2.2. If they answer correctly, print "Correct!" and add 1 to their `score`. Otherwise print "Incorrect!" and the correct answer.

- *The "`sum()`" function can be used to determine the correct answer.*
- *Convert the user's input to an integer so that you can properly compare it to the correct answer.*

2.7. Otherwise, ask the user "What is the average of the numbers in this list?" and display the list of numbers generated in Requirement 2.2. If they answer correctly, print "Correct!" and add 1 to their `score`. Otherwise print "Incorrect!" and the correct answer.

- *Also print a "round UP to nearest integer" message.*
- *There are various ways to determine the correct answer (remember to round it up to the nearest integer). The "`sum()`", "`len()`" and "`math.ceil()`" functions are likely to be useful.*
- *Convert the user's input to an integer so that you can properly compare it to the correct answer.*

3. After the loop (Requirement 2) ends, print a "test complete" message, followed by the user's `score` out of `questions`, as well as the percentage that the user's score represents, in the following format: `You scored 3/4 (75.0%).` If all questions were answered correctly, also print "Perfect score, well done!"

- *Round the percentage to 1 decimal place.*

Tip: Check the Canvas Announcements/Discussion on a regular basis – extra tips, examples and advice regarding the assignment will be posted there throughout the semester. You are also welcome to make your own posts if you have questions – but don't upload your work to the discussion board!

The "random_list" Function

You must create a function named "`random_list()`" and use it to implement Requirement 2.2. The definition of the function should be at the start of the program, and it should be called where needed in the program. The function requires three parameters (all integers) named "`quantity`", "`minimum`" and "`maximum`". The function must generate and return a list of `quantity` random integers between `minimum` and `maximum` (inclusive).

For example, "`random_list(5, 1, 10)`" would return a list of 5 random integers between 1 and 10, such as [4, 2, 5, 10, 2]. There are various ways to achieve this, but most approaches are is likely to involve a list, a `for` loop, and the "`random.randint()`" function.

Remember to ensure that the function receives the `quantity`, `minimum` and `maximum` as *parameter values*, and *returns* the resulting list. The function should not reference any global variables or print anything. Revise Module 4 if you are uncertain about these concepts.

Ensure that the function does exactly what is specified above and nothing more – it is important to adhere to the stated specifications of a function, and deviating from them will impact your marks. Create *additional functions* (2 max) in your program to achieve a modular solution.

Additional Features (Implement these additional features on top of basic program requirements in a separate "main_additions.py" file)

Below are some additional features that you can make to the program to further test and demonstrate your programming ability.

- (1 point) When prompting the user to select a difficulty, it allows "easy", "medium" and "hard" as well as just the first letter of the word. Ensure that it is *not* case-sensitive.

- (1.5 point) Time how long it takes the user to answer each question. This can be achieved by importing the "time" module and using the "`time.time()`" function, which returns a float representing the current time in seconds. Get the time just *before* prompting the user to answer a question, and subtract it from the time just *after* you get the user's input. Round the result to 1 decimal place and display it with the "correct/incorrect" message, e.g. "Correct! You answered in 2.1 seconds".

- (1 point) Ensure that your program does not crash if the user enters something that is not an integer when prompted for an answer to a question. Instead, your program should show an "invalid input" message and prompt the user again until they enter something valid. This is best done using exception handling (Module 6), however what you need to know is covered in Workshop 4 – create the input validation function and use it to prompt the user for input.

- (1 point) Include an additional question type – "What is the difference between the smallest and biggest numbers in this list?" e.g. The answer would be 8 in a list of [5, 12, 8, 4, 6], since that is the difference between 4 and 12.

- (1.5 point) Include another additional question type – "How many #s are there in this list?", with "#" being replaced by one of the numbers that exists in the current list. The "`random.choice()`" function can be used to select a random value from a list, and the "`.count()`" list method can be used to determine how many times a value exists in a list.

Submission of Deliverables

Once your assignment is complete, submit both your **pseudocode** (in PDF format) and **source code** ("main.py" and "main_additions.py" files) to the appropriate location in the Assessments area of Canvas. *Zipping the files is not required.* An assignment cover sheet is also not required, but be sure to **include your name and student number at the top of both files (not just in the filenames)**.

Marking Key

Criteria	Marks
Pseudocode These marks are awarded for submitting pseudocode which suitably represents the design of your source code. Pseudocode will be assessed on the basis of whether it clearly describes the steps of the program in English, and whether the program is well structured.	**5**
Functionality These marks are awarded for submitting source code that implements the requirements specified in this brief, in Python 3. Code which is not functional or contains syntax errors will lose marks, as will failing to implement requirements/functions as specified.	**14**
Additional Features These marks are awarded for submitting source code that implements the additional features specified in this brief, in Python 3. Code which does not include these additional features, is not functional or contains syntax errors will lose marks.	**6**
Code Quality These marks are awarded for submitting well-written source code that is efficient, well-formatted and demonstrates a solid understanding of the concepts involved. This includes appropriate use of commenting and adhering to best practise.	**5**

Total:	**30**