University of Westminster
School of Computer Science & Engineering

4COSC001W: Programming Principles I - Assignment Specification (2018/9)	
Module leader	Wendy Purdy
Unit	Practical Exercises
Weighting:	50%
Qualifying mark	30%
Description	Practical Work
Learning Outcomes Covered in this Assignment:	The coursework rationale is: • LO1 Identify program requirements and select appropriate algorithms to implement them; • LO2 Represent algorithms in a structured manner (e.g., by use of flow diagrams or pseudo-code); • LO3 Implement algorithms using an algorithmic, strongly typed programming language, and design and run appropriate tests on the resulting code; • LO4 Write program code that conforms to norms of good style and meets generally accepted referencing criteria;
Handed Out:	22th October 2018
Due Date	**Tuesday 20th November at 1:00 PM**
Expected deliverables	1. Part 1 - Assignment (Histogram). Flow chart, Python program code, test case results. 2. Part 2 - Assignment (Encoding / Decoding Strings). Python program code. 3. Viva - Monday 26 November (during your scheduled tutorial)
Method of Submission:	Submitted online via Blackboard
Type of Feedback and Due Date:	Written feedback and marks 15 working days (3 weeks) after the submission deadline. All marks will remain provisional until formally agreed by an Assessment Board.
BCS Criteria covered in this Assignment:	2.1.1 Knowledge and understanding of facts, concepts, principles & theories 2.1.2 Use of such knowledge in modelling and design 2.1.3 Problem solving strategies 2.2.1 Specify, design or construct computer-based systems 2.2.4 Deploy tools effectively 2.3.2 Development of general transferable skills 4.1.1 Knowledge and understanding of scientific and engineering principles 4.1.2 Knowledge and understanding of mathematical and statistical principles

Assessment regulations
Refer to section 4 of the "How you study" guide for undergraduate students for a clarification of how you are assessed, penalties and late submissions, what constitutes plagiarism etc.

Penalty for Late Submission
If you submit your coursework late but within 24 hours or one working day of the specified deadline, 10 marks will be deducted from the final mark, as a penalty for late submission, except for work which obtains a mark in the range 40 – 49%, in which case the mark will be capped at the pass mark (40%). If you submit your coursework more than 24 hours or more than one working day after the specified deadline you will be given a mark of zero for the work in question unless a claim of Mitigating Circumstances has been submitted and accepted as valid.

It is recognised that on occasion, illness or a personal crisis can mean that you fail to submit a piece of work on time. In such cases you must inform the Campus Office in writing on a mitigating circumstances form, giving the reason for your late or non-submission. You must provide relevant documentary evidence with the form. This information will be reported to the relevant Assessment Board that will decide whether the mark of zero shall stand. For more detailed information regarding University Assessment Regulations, please refer to the following website: **http://www.westminster.ac.uk/study/current-students/resources/academic-regulations**

Coursework Description

Part 1 - Histogram for Exam Marks

The University requires a graphical display to show how many students received different grade categories for a module assessment. You are required to write a program in Python that achieves this.

- Flow Diagram - Note that before you start to program your solution you should create your flow diagram that represents your algorithm in a structured manner.
- A set of test cases will be provided for you to test your program solution.

Part A: Basic Program

- The program should allow the tutor to enter in the various marks awarded, until the tutor enters in a negative mark (e.g., -1).
- When a negative mark is entered, the program should display a histogram where each star represents a student who achieved a module mark in the category range: 0-29, 30-39, 40-69 and 70-100.

This example shows the output distribution for 20 marks. Your program should work with any number of marks entered.

```
0-29      ***
30-39     *****
40-69     ********
70-100    ****
20 students in total.
```

- A counter should count the number of student's marks which have been entered.
- The display should be neatly formatted.
- Make use of 'loops' for the display of the stars for each category.

Part B: Statistics
Extend your program so that the following statistics display after the histogram.

a) Average mark.
b) Number of students with a pass mark (mark of 40 or above).
c) Highest mark
d) Lowest mark.

Part C: Input Validation
Valid marks are integers 0 to 100.

- Your program should display an appropriate message if a non-numeric character is entered.
- Your program should display an appropriate message if a number over 100 is entered.

(Optional) Part D: Vertical histogram (Extension)
Should only be attempted if you are confident with your programming skills. No tutor help will be provided.
The part 1 histogram shows each category horizontally across the screen. Extend your program to add an additional histogram that displays vertically so the stars in a category should go downwards and not across the screen, e.g.:

```
0-29   30-39   40-69   70-100
  *      *       *       *       (as a line is printed decide if each category needs a star or space)
         *       *
         *
```

Test Cases
Test your solution using the test plan specified and complete with your final results (pass, fail or not attempted). Submit the test plan (with your code) and bring a printed copy of your submitted test cases to the viva. Marks for the test cases can only be allocated if you attend the scheduled viva.

Part 2 - Encoding / Decoding Strings

This assignment will involve the use of strings and functions.

Create a Python program that will apply a simple encoding and decoding of a message.

The program should prompt the user to enter a message and then enter a key (the shift value). The key should be in the range 1 to 25. The program will then produce an encoded string by applying the key (shift value) to the alphabetic characters in the message. The shift amount instructs the program on how many places in one direction to move to find the encoding of any letter of the alphabet.

Examples: A key of 3 will shift each character 3 characters to the right.

- the character '**a**' as the character '**d**',
- the character '**b**' as '**e**', etc.,
- the string **'this'** becomes the string **'wklv'** using a rotation of 3.

Wrap the encoding around the alphabetic characters, so that 'w' is encoded as 'z', 'x' is encoded as 'a', 'y' is encoded as 'b', and 'z' is encoded as 'c' etc.

Important note:

- There are various solutions to this type of challenge on the internet. These solutions convert a letter to ASCII code using ord() and convert an ASCII code to a letter using chr(). Instead your solution should be built around rotating a character in string. E.g.,

  ALPHABET = 'abcdefghijklmnopqrstuvwxyz'

- Any code solutions submitted using the ord() and chr() approach will receive a mark of zero.

Part A: Encoding
The program prompts for a string to encode and a rotation integer in the range of 1-25. The program then returns the encoded string. **Important**, the program should not encode any letter that is not in the lower case alphabet. Those letters should simply be passed through to the encoded string. E.g.,

- the string **'*this!'** becomes the string **'*wklv!'** using a rotation of 3.

(Optional) Part B: Decoding

- The program should prompt the user for one of three commands:
  - 'e' to encode a string
  - 'd' to decode a string
  - 'q' to quit

- If the command is encode, then the program runs as for Part A: Encoding.
- If the command is decode, then the program should prompt for a string to decode and key and the program displays the encrypted/decrypted message.
- Any other command should raise an error and reprompt.

(Optional) Part C: Decoding (Extension)
Should only be attempted if you are confident with your programming skills. No tutor help will be provided.
If the command is decode, then the program should prompt for a string to decode and a plain-text word that appears in the text (decoded string). The output should be the rotation needed to decode the string and the decoded string. Example:

If the encoded string is: *khoor zruog* and the plain-text word is: *hello*
The program should work out the rotation was *3* and the decoded string is '*hello world*'

Part 3 - Compulsory Viva (for Part 1 and Part 2)

The viva will be during your scheduled tutorial on 26 November. Demonstrate your working solution to your tutor by running the code. Bring a printed copy of your submitted test cases to the viva for Part A.

In addition

- Use descriptive variable names and comments to document your code.

- Ensure you meet accepted referencing criteria for any code modified from other sources.

Marking scheme

The Coursework will be marked based on the following marking criteria:

Criteria	Marks per subcomponent	Mark per component
Part 1 - Assignment (Histogram)		55
Flow diagram of algorithm solution	10	
Part A: Program - marks entered & horizontal histogram Part B: Statistics displayed Part C: Input validation Part D: (Optional Extension) Vertical histogram	16 4 4 6	
Test case results (results match code submitted & demonstrated at viva)	15	
Part 2 - Assignment (Encoding / Decoding Strings)		30
Part A: Program accepts a message and a key (1 to 25). Encodes lower case characters only	12	
Part B: (Optional) Menu of three commands. Decrypts using encrypted message and key	6	
Part C: (Optional Extension) Decrypts using encrypted message and word from decoded string	6	
User-defined functions	6	
Part 3 • Programming style & Acceptable referencing • **Viva:** Ability to answer questions about solution o Part 1 o Part 2		15
Total:		100