CSP1114 – ASSIGNMENT (50%)

LO2: Choose appropriate digital tools for program design and problem solving.

DETAILS:

1. This assessment carries 50% of your total coursework marks for this course.
2. Form groups of 2 to 4 students.
3. Write a Python application that is engaging, unique, and impactful with interesting or entertaining functionality. The key is that the program should be fun and enjoyable to interact with in some way. Before starting, you will need to propose your program title and features to your lecturer. Please consult your lab lecturer for guidance on how to submit your program proposal.
4. Your program must include the following:
    - Variables names appropriately.
    - Selection structures.
    - Loop structures.
    - Functions that you write yourself.
    - The ability to save data into files.
5. Remember that this is an assignment for at most FOUR students, and the syllabus allocates 21 hours of independent learning time this trimester for you to work on it. As such, the complexity of the program should reflect this. Please **READ THE RUBRICS** to see how you will be scored.
6. We are expecting you to build a **text-based** console program.
7. Examples of the kinds of programs you can build include the following. You are NOT limited to these, they're merely ideas to get you started:
    - Quizzes / Word Puzzles based on Maths / Physics / English / other subjects
    - Text-based Game Show Simulations (Wheel of Fortune, Hangman, Wordle)
    - Text-based Adventure Games
    - Children's Information Based Games / Puzzles
8. Following a tutorial online is **NOT** sufficient work, as copying and running code you see in tutorials is **NOT** proof of learning to code. If you wish to use a tutorial, use it to start with a base game, and then modify the game by adding things like leaderboards, more functionality, different puzzles, making combat more complicated, etc. (Adding levels usually teaches you nothing.) You should be the one adding the items stated in (4) above. When you want to base your assignment off a tutorial, do these two things:
    - State clearly in your submission (comment in the source file) your source (link etc).
    - Include a separate source file with the program the tutorial teaches you to build BEFORE MODIFICATIONS. Your lecturer can then compare the tutorial's code vs your final assignment submission and get a clear idea of what you added and your own work.

    If you do not state the source of your work and we find a tutorial online for a game similar to the one you submitted, you will receive a **ZERO** for your assignment.
9. Make your code readable through well-named variables and functions. Document complicated/difficult parts using comments.
10. Your source code must be able to be interpreted without errors with Python 3.11 Interpreter or later running on a Windows machine. If your source code cannot be interpreted (run), you get **ZERO** for your whole assignment.

WORK DIVISION:

At your level as beginners learning coding, and due to the nature of this assignment, all members of your group will likely be working on the same set of files. As such, a good way to divide work is to first figure out the features of your program and divide out the work accordingly. An example follows:

- 1 student works on the main sequence and interface of the program.
- 1 student works on functions that provide functionality A.
- 1 student works on functions that provide functionality B.
- 1 student works on functions that provide functionality C.

By keeping your work within functions, you can write clean, self-contained code which can be combined with another student's work more easily. There is also a higher possibility of working simultaneously rather than waiting for one person to finish a portion before continuing to work on a different portion. Of course, it is possible that you will need some baseline code ready before others can add or build on top of it. Please start working on the project early as it is impossible to code last minute, especially for a group project!

Although the project will be marked as a group, your individual score may be adjusted based on your ability to *explain your own code and contributions* during the assignment interview, as well as *the flowchart* for the features you worked on. (See the rubrics for details.)

USING GIT (***OPTIONAL***)

Git is a great tool for collaborative programming and keeping track of who contributed/changed each line of code. Regular commits through git is the best way to prove to your lecturer your contribution and the work division in your group. It is beyond the scope of this course and is **OPTIONAL**, but it is something worth learning and using for ALL your future group programming projects.

- Video – What is Git: https://www.youtube.com/watch?v=hwP7WQkmECE
- How to install: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- How to use: https://www.w3schools.com/git/
- Using it with GUI (if command line confuses you): https://www.sourcetreeapp.com/

DELIVERABLES:

Your final deliverable should be:

- Flowchart file from each individual group member in PDF format. Please name the PDF file as "**YourName.pdf**".
- Python ".py" file containing the source code of your program. Optional: Any other necessary assets in subfolders. If you have more than one file, put them in a folder and zip it up. Submit the zip file.

Your source code MUST start with the following:

```
# ****************************************************************************
# Program: YOUR_FILENAME.py
# Course: CSP1114 PROBLEM SOLVING AND PROGRAM DESIGN
# Lecture / Lab Section: TCXL / TLXL
# Trimester: 2430
# Names: MEMBER_NAME_1 | MEMBER_NAME_2 | MEMBER_NAME_3 | MEMBER_NAME_4
# IDs: MEMBER_ID_1 | MEMBER_ID_2 | MEMBER_ID_3 | MEMBER_ID_3
# Emails: MEMBER_EMAIL_1 | MEMBER_EMAIL_2 | MEMBER_EMAIL_3 | MEMBER_EMAIL_3
# ****************************************************************************
```

DEADLINE:

- Submission in eBwise: **7 February 2025 11.59 pm (Week 14).**
  - Only one person should submit the program file. Each individual member is responsible for submitting their own flowchart file**.**
- Code interview: **During study week**
  - You will be interviewed to justify your contribution to your group's work.
  - Please consult with your lecturer if you wish to submit your work and be interviewed earlier, before study week.

PENALTIES:

- Penalty of 1 mark for each day of late submission
- We will give ZERO (0) marks to students who plagiarise AND to students who intentionally or unintentionally help other students plagiarise by giving all or some of their code.

SCORING RUBRICS:

The student will be marked based on the rubric below. Final formula:

$$Final\ Score = \frac{Assignment\ Score}{24} \times 50 \times \frac{(Q\&A + Flowchart)}{8}$$

INDIVIDUAL SCORE

Criteria	4 – Is Own Work	3 – Some Doubts	2 – Many Doubts	1 – Not Own Work
Q & A	Student able to answer all interview questions. There is no doubt in the lecturer's mind that the program is the student's own work.	Student unable to answer some interview questions. There are segments of the program the student is unable to explain.	Student unable to answer most interview questions. Student unable to explain many parts of the program.	Student unable to provide even a general idea of how the program works program is clearly not student's own significant effort.

Criteria	4 – Excellent	3 – Good	2 – Satisfactory	1 – Needs Improvement
Flowchart	The flowchart is highly clear, logically organized, and correctly represents the entire program flow. All steps are appropriately labelled, with well-defined decision points and processes. Arrows are used correctly to indicate the flow of control, and there are no ambiguities or redundancies.	The flowchart is mostly clear and organized, with 1 to 5 errors (e.g., missing or misconnected steps, incorrect labels, unclear decision points). The overall structure and logic are easy to follow, though some minor issues are present.	The flowchart includes most of the necessary steps and decision points, but there are some ambiguities, missing elements, or unclear connections - contains between 6 and 10 errors. The logic is mostly correct, but may require clarification	The flowchart has more than 10 errors, such as missing critical steps, incorrect connections, or major logical issues. The design is difficult to follow, and the flowchart requires significant revision to be usable.

Note: If one member's flowchart is more complex than the others (e.g., containing more control structures), the rubrics may be adjusted based on its complexity.

ASSIGNMENT SCORE

Criteria	4 – Exceeding	3 – Meets	2 – Approaching	1 – Does Not Meet
Requirements	The student coded advanced constructs **beyond what is required**. (Objects / data structures / lambda / advanced external libraries, etc)	The student **coded all the required constructs**: variables, selections, loops, functions, and file management.	The student **coded 4 out of the 5 required constructs**.	The student **coded 3 or less of the required constructs**.
Difficulty	Writing the program requires additional **knowledge outside of what is taught in the class**. Demonstrates **self-learning** in programming.	In writing the program, the student **makes use of knowledge taught in class in more depth**. Demonstrates deep learning of programming.	In writing the program, the student mostly **reproduced the examples shown in class**. Demonstrates surface learning of programming.	The program used **examples from class poorly or made use of limited programming**. Demonstrates lack of understanding in programming.
Interaction and Output	The program provides **smooth and intuitive interaction** with the user. All outputs are accurate, well-formatted, and meaningful	The program allows for **correct interaction, and outputs are mostly accurate and well-formatted**. Some minor issues with input handling or output clarity.	**User inputs offer no meaningful choices**. (E.g. Only need to click 'next' to move to the next output.) Output is neat and easy to follow.	The program requires **no user input. Runs automatically** like a slide show or cutscene. Output is messy or hard to follow.
Originality, Engagement, Uniqueness, and Impactfulness	The program excites the user and has a 'wow' factor. **Highly original, offering a fresh and unique experience**. It is engaging, entertaining, and interactive, with functionality that keeps the user interested and immersed.	The program is mostly **functional and entertaining**. Offers some level of engagement. It includes interactive or entertaining elements that are enjoyable, though the experience may not be as immersive or impactful as excellent work.	The program is functional, and some parts of it are quite fun to use. It is somewhat engaging and functional but **lacks the depth or creativity to keep users fully entertained**.	The program lacks originality or creativity, relying on generic or predictable design. It **does not engage or entertain the user, and its functionality feels basic or uninspired**. The overall experience is not fun or enjoyable.
Error Free Execution	The program **runs without errors for expected input, recognizes wrong input and notifies the user**.	The program **runs without errors for expected input, or no possible input errors.**	The program **runs with some errors, but user is still able to see it in action till the end**.	**Errors make the program unusable** and disallow the user from seeing the full extent of the program.
Organisation	Naming convention and comments sufficiently makes the code easy to read throughout. Work division is documented through comments OR used Git to track commits and member contributions.	Naming convention and comments sufficiently makes the code easy to read throughout.	Naming convention and comments are used in critical segments of code, but there is some confusion in other segments of code.	The naming conventions and comments are poorly used, making the code difficult to read.