Unit 1 : Principles of Computer Science : Python Programming Project : Tic-Tac-Toe

---

Tic-tac-toe (also known as noughts and crosses or Xs and Os) is a game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

The following example game is won by the first player, X:

https://en.wikipedia.org/wiki/Tic-tac-toe

Your task:

Plan, develop, test and evaluate a Python implementation of this game
All your work must be handed in as a github repo. Instructions and information should be in markdown, png or pdf files, and all code and text files used must also be handed in.

You may develop your code in any jupyter notebooks or any other industry standard IDE

Plan

You must first plan your solution

1. **RULES**: Play the game a few times to fully understand the rules. Write the rules, simply, in your own words
2. **PLAN**: Make notes about what decisions you will need to make in your program
    a. Variables and constants and their types
    b. Use of data structures and files
    c. User interface - how the user will interact with the program. The messages they will receive.
3. **ALGORITHM**: Design the algorithm as a flowchart
4. **TEST PLAN:** Create a comprehensive test plan using the following template:

Test Num	Description of Test	Test data	Expected outcome
1	I will input normal test data	valid	The program will run
2	I will attempt to input false or incorrect information	erroneous	The program should crash

PLANNING	Total For Task = 10 Marks

Develop

You must now code your solution for the tic tac toe game that you have planned.

You must hand in at least 3 distinct versions of your code. Each must be a fully functional Minimal Viable Product (MVP) with just enough features to satisfy early users, and to provide feedback for future product development. (https://en.wikipedia.org/wiki/Minimum_viable_product)

1. **DEVELOPMENT DIARY and CODE:** For each stage in development (each MVP) you should write up
    a. What features are being developed
    b. Show the code and annotate with comments or other
    c. Show testing for functionality and debugging

DEVELOP	Total For Task = 14 Marks

Test

Once your code is complete, go back to your test plan and add 2 more columns:

Actual Outcome	Comments and fixes

Now run each test with the final version of your game, adding comments where the actual outcome differs from the expected outcome. Fix errors if possible, discuss errors if not.

TEST	Total For Task = 7 Marks

Evaluate

Evaluate your program according to the following:

- The coding conventions that you used
    - Identifier naming conventions
    - Use of external libraries/modules
    - Decomposition of code - use of functions and parameters
- The quality of your solution for the user

    - How easy it it to use and play?
    - What features does it provide?
    - Can the user crash? Did you take unexpected events into account and write
- The quality of your code for programmers
    - How easy it it to read?
    - How easy it is to modify or extend?
    - Is the code robust? (Can the user crash the program?)
    - Use of memory - global and local variables, use of functions and parameters
    - The performance of your program - how efficient is the code? Where could it be improved?

EVALUATE	**Total For Task = 6**
Marks	

TOTAL	**Total For Task = 36 Marks**

Assessment Grid

PLAN	1-3	4-6	7-10
Rules **Plan** **Algorithm** **Test Plan**	Rules are missing or are not logical and clear Planning of data and UI is missing or very sketchy Flowchart is missing, not complete or had many errors including using incorrect symbols Test plan is too narrow to confirm a working solution including limited normal, abnormal and/or extreme data. Expected results are generic or mostly inaccurate based on identified test data.	Rules are adequate although not fully or clearly described Some planning of data and UI has been done but there are omissions or errors Flowchart is attempted, symbols are mostly used appropriately, most component parts are shown including inputs and outputs, although errors may exist. Test plan is adequate to confirm a working solution, including some normal, abnormal and extreme data. Expected results are and accurate based on identified test data, but may lack detail.	Rules are fully and clearly described Data used within the solution and well as the UI is clear and well-planned flowchart symbols used accurately throughout. Flowchart breaks down requirements into component parts that are detailed and relevant. The flowchart shows full coverage of inputs, outputs and processes using naming conventions appropriate to the scenario consistently. Links between component parts are complete and efficient with accurate and robust procedures for handling unexpected events Test plan is thorough, including a range of normal, abnormal and extreme data. Expected results are specific and accurate based on identified test data.

DEVELOP	1-5	6-10	11-14
MVP1 **MVP2** **FINAL**	Limited use of accurate syntax and indentation appropriate for the chosen language. Organisation has structure that lacks logic and commenting is vague, making maintenance of the code by a third party difficult. Code is inefficient; uses limited appropriate and accurate programming conventions. Uses imprecise logical operations to create a program which may not function or compile and/or may have major errors that prevent the program from meeting the given criteria. Program outputs may contain inaccuracies and/or provide limited information so a novice user would experience difficulty in using the program. Program uses minimal validation and checking procedures resulting in a program with limited capacity to reduce errors or handle unexpected events.	Program uses mostly accurate syntax and indentation throughout, appropriate to the chosen language. Organisation has logical structure and commenting is informative, but not always clear, allowing for the code to be maintained by a third party. Code is efficient; uses appropriate and accurate programming conventions throughout. Uses logical operations with some precision to create a functional program that meets the given criteria with minimal errors. Program outputs are accurate and mostly informative allowing a novice user to use the program. Program uses accurate validation and checking procedures, resulting in a program that minimises errors and handles unexpected events.	Program uses accurate **syntax** and indentation throughout Organisation has **logical structure** and **commenting** is consistently clear and informative, allowing for the code to be easily maintained by a third party. Code is highly **efficient** and optimised; uses appropriate and **accurate programming conventions** throughout. Uses precise logical operations throughout to create a **fully functional, error-free program** that meets the given criteria. Program **outputs are accurate and informative** allowing a novice user to easily use the program. Program uses **accurate validation and checking** procedures throughout, resulting in a **robust** program that minimises errors and handles unexpected events.

TEST	1-2	3-4	5-6
Testing	Testing shows evidence of a limited or linear development process, with minimal identification and resolution of errors. Comments show a limited understanding of errors that were found, and how they were fixed.	Testing shows evidence of an iterative development process that identifies and resolves some errors, but problems may persist. Comments show partial understanding of errors that were found, and how they were fixed.	Testing shows evidence of an iterative development process that identifies and resolves errors and improves efficiency. Comments show a clear and detailed understanding of errors that were found, and how they were fixed.

EVALUA-TION	1-2	3-4	5-6
	Superficial understanding of relevant technical concepts with some inaccuracies. Limited or unsupported justification of changes made during the development process. Limited or unsupported justification of coding conventions selected. Limited links between aspects of the solution and the requirements of the scenario. Limited or unsupported judgments about the quality and performance of the program. Technical vocabulary is used but it is not used appropriately to support arguments.	Some accurate and relevant understanding of technical concepts. Some valid justification, which may lack support, of changes made during the development process. Some valid justification, which may lack support, of coding conventions selected. Some logical links between aspects of the solution and the requirements of the scenario but may lack clarity. Some valid judgments which may lack support about the quality and performance of the program. Mostly accurate technical vocabulary is used to support arguments.	Accurate and detailed understanding of relevant technical concepts throughout. A valid and fully supported justification of changes made during the development process. A valid and fully supported justification of coding conventions selected. Makes logical coherent links between aspects of the solution and the requirements of the scenario throughout. Makes valid and fully supported judgments about the quality and performance of the program. Fluent and accurate technical vocabulary is used to support arguments.