# Portfolio (POR) Assessment Part 1

**Assessment Resources:**

|  |
| --- |
| Marking key available for lecturer via Blackboard.  Students may refer to the lecture material in formulating their answers.  Material for this assessment is available:  [*https://github.com/NM-TAFE/pin-civ-ipos-por1-preview*](https://github.com/NM-TAFE/pin-civ-ipos-por1-preview) *NOTE: If you were invited to use GitHub classroom by your online lecturer, you can use the link above as a preview but* ***please use the GitHub classroom link provided for your work.*** |

**Assessment Instructions:**

|  |
| --- |
| Students must attempt all questions. Answer succinctly using full sentences. At most two paragraphs are expected per answer.  All answers must be at the student’s own words – copying generated code or answers from ChatGPT or other AI tools is **strictly** prohibited.  **Please** **note**: you may have been invited to trial completing this assessment via GitHub classrooms. If you have, you do not need to also complete this assessment. |

# Overview

Tic-Tac-Toe Game Refactoring

## Objective

Refactor a given monolithic tic-tac-toe game, such that the code:

1. Is modular, consisting of at least two files that logically group related functions.
2. Implements an appropriate Python project folder structure.
3. Includes at least one test case.
4. Employs a 2D data structure.

In the process, you must use at least four functions, two classes, two files, and one import statement of your modules (not including imports used in the test case).

The source code you need to refactor can be found here:  
  
<https://github.com/NM-TAFE/civ-ipos-sessions/tree/2023S2/assessments/POR/por-1>

## Instructions

Complete each of the following steps carefully reviewing what is expected.

### Step 1: Review the Existing Code

Firstly, analyze the given tic-tac-toe game code. Understand the flow and functionality before proceeding with the refactoring.

### Step 2: Identify Components to Refactor

Identify the parts of the code that can be improved. Determine which parts of the code can be grouped *logically* into separate modules.

### Step 3: Modularizing the Code

Refactor the code to create at least **two** files. These files should contain logically grouped functions and/or classes. Ensure the file names are appropriate for the division you have chosen.

### Step 4: Create a Modern Python Folder Structure

The refactored code should adhere to the following Python folder structure:

tic\_tac\_toe/

|--- src/

| |--- \_\_init\_\_.py

| |--- module1.py

| |--- module2.py

|--- tests/

| |--- \_\_init\_\_.py

| |--- test\_tic\_tac\_toe.py

|--- setup.py

**Note**:

* \_\_init\_\_.py files are used to indicate that a directory should be treated as a Python package. This allows the files within to be imported as a module in the test scripts or other python files.
* setup.py is a Python file used to specify what modules and dependencies must be installed. The file has been provided for you, along with instructions on how to install your modules using this file. If PyCharm prompts you to run the setup.py – cancel – it will not work until you restructure your project!
* You must give your Python files appropriate names. Do **not** use module1, 2, etc.

### Step 5: Create a Test Case

Develop at least one test case for your refactored code. The test case should reside in the 'tests' directory. It is preferred that you use unittest framework for writing your test case.

### Step 6: Implement 2D Data Structure

Refactor the code such that it employs a 2D data structure for the tic-tac-toe game board.

### Step 7: Written Report

Once you have completed your refactoring, write a brief report addressing the following:

1. Justification for your refactoring decisions.
2. The challenges you would have faced maintaining and testing the original monolithic code.
3. How you would modify your refactored code to handle a custom-sized tic-tac-toe game (larger than 3x3), and how this implementation would be easier to handle than in the original code.

<<Space for your answer>>

In the refactored code, we just have to change a parameter of the game and we have a board larger than 3x3. All function are dependant on the board size (check wins, check full, check valid positions, etc…)

Whereas in the monolitic code we would have to go through all the code and check / adapt for the impact of a larger game.

Justification

Challenges

Approach to adding support for larger boards

### Step 8: Short Answer (Knowledge Questions)

Provide brief answers to the knowledge-question worksheet.

Briefly explain what is modular programming.

Modular programming is the concept to split a program into different modules. Each modules must be independent, and contain its own class and function to manage its limited scope

How can you import only a specific function or class from a module in Python? What is the syntax for this?

from ‘moduleX’ import ‘classX or functionX’

How would you explain Python's parameter-passing mechanism? Is it more like pass-by-value or pass-by-reference? Justify your answer.

Python is pass-by-reference. It means that a variable is a reference to an object.

If a variable points to a mutable object (ex : list) and a function modifies that object, the modification is done directly to the object, so if we call this variable again later on we would see that modification.

Which is different from pass-by-value programming language, where a function received a copy of the object, and modifications are done to the copy and not the initial object. So outside of the function, the modification is not seeable.

Given the following Python code, what will be the output and why?

def modify\_list(list\_):

list\_.append("new")

list\_ = ["completely", "new"]

items = ["original"]

modify\_list(items)

print(items)

# Output:

["original", "new"]

# Explanation:

The function first modifies the list (append).

Then the variable list\_ within the function is re-appointed to another list. But the ‘original’ variable is not reappointed to anything.

So the function has only append “new” to the ‘original’ list, hence the output.

In Python, even though variables created within a function are local, there are still situations where you can modify data outside the scope with a local variable. Explain this anomaly and relate it to both mutability and pass by reference.

Python is a pass-by-reference language. Meaning that a pointer to the original variable is passed to the function.

For immutable objects (like integer) if a variable is changed during a function, the pointer is reallocated to another object (because integer is immutable). So changes done are not visible outside of the function

For mutable objects, as the function uses a pointer to the objects, the object is modified, and the modification remains outside of the function.

List two benefits of modular coding approaches. How do these benefits assist in the development of medium-sized applications?

Reusability: Once a module is written, it can be reused across the program, or even in other programs. This saves time and effort and promotes code consistency.

Maintainability: When functionality is separated into modules, it becomes easier to maintain and update the software. If a change needs to be made to a specific feature, you only have to modify the related module, minimizing the risk of unintentionally affecting other parts of the program.

Testing and Debugging: It's easier to test and debug modules as standalone units, and any bugs that are found and fixed in a module will be fixed everywhere the module is used.

Collaboration: In large projects, different teams can work on different modules simultaneously with less risk of interfering with each other's work."

### Submission

Please submit the refactored code including the root folder but **excluding** the venv/ via Blackboard, also attach this worksheet. Note: if you are trialling the new GitHub classroom approach you only need to submit a note specifying that you completed the assessment.

Name your zip and document:

XXX\_source-por-part1.zip and XXX\_C-IPOS-AT1-POR-Part1.docx respectively.

Where XXX represents **your** two or three-character initials.

### Evaluation Criteria

Your refactoring will be evaluated on the clarity and modularity of your code, as well as the thoughtful reasoning behind your design decisions. Your test case should be robust and cover key aspects of the tic-tac-toe game functionality. The written report should accurately reflect your understanding of code refactoring, testing, and the flexibility of your new implementation.