Government of Western Australia
North Metropolitan TAFE

# Portfolio Assessment Task 4

| | |
|---|---|
| **Qualification national code and title** | ICT50220 Dip Advanced Programming |
| **Unit/s national code/s and title/s** | ICTPRG535 – Build advanced user interfaces<br>ICTPRG547 – Apply advanced programming skills in another language |

## Assessment type (☑):

☒ Questioning (Written)

☐ Practical Demonstration

☐ 3rd Party Report

☒ Portfolio

## Assessment Resources:

Python3 interpreter.

Python IDE, like PyCharm, or VSCode (the latter is not supported by the college), with the ability to use Python Virtual Environments.

Access to Office 365 and Microsoft Word.

Git and access to GitHub.

*Use of some of these items may not occur in this part of the assessment task.*

## Assessment Due

This item is due:

- **Week 12, 17:00 (5pm) on the day of the scheduled lecture**

Refer to Blackboard for most accurate dates, which may alter due to unforeseen circumstances. We also endeavour to update these documents at the same time.

*It is advantageous for you to attempt to meet this deadline.*

# Portfolio Assessment Task 4

| | |
|---|---|
| **Qualification national code and title** | ICT50220 Dip Advanced Programming |
| **Unit/s national code/s and title/s** | ICTPRG535 – Build advanced user interfaces<br>ICTPRG547 – Apply advanced programming skills in another language |

## Assessment Instructions:

### Scenario

You are employed as a junior software developer at a boutique software house called *Softwares-R-Us* in Perth.

You have been assigned to the Games team and will provide some of the code required for various games that the company builds and releases.

You will be updating the existing code from the previous tasks to search a player by their name in a list of `Player` objects.

### Instructions

Your code must adhere to certain style guides, the most important being PEP-8 – Style Guide for Python Code. You should familiarise yourself with PEP-8 before continuing.

Use of a Git repository is standard practice at *Softwares-R-Us* and most of the steps require the use of Git and GitHub.

There are multiple steps in this task, and you must perform each step to a satisfactory level.

You may answer any questions in the provided template (if available) and the use of screenshots is encouraged. However, you must also provide the actual source code as a ZIP-file of the project for any programming tasks. *Make sure to remove the Virtual Environment folder (*`venv` *or* `.venv`*) from the ZIP-file before uploading.*

You must document your code properly. Use docstrings where needed including but not limited to entire classes and methods. Use inline comments to clarify certain parts of code.

If you use any external resources, you should provide references.

# Portfolio Assessment Task 4

| | |
|---|---|
| **Qualification national code and title** | ICT50220 Dip Advanced Programming |
| **Unit/s national code/s and title/s** | ICTPRG535 – Build advanced user interfaces<br>ICTPRG547 – Apply advanced programming skills in another language |

## Assessment Instrument:

### Step 1 – Knowledge Question (20-50 words)

In your own words, describe what a Binary Search Tree (BST) is.

In addition, describe two important properties of a BST: depth and height. How are they different?

### Step 2 – Knowledge Question (50-80 words)

In your own words, describe how an algorithm to find an item in a Binary Search Tree works.

### Step 3 – Knowledge Question (20-60 words)

In your own words, describe what a balanced BST is.

### Step 4 – Prepare Binary Search Tree

In this step, you will create the framework for a Binary Search Tree of Player objects. This step has multiple parts to it:

a. Add a new file to the folder `app` called `player_bst.py`. This will contain the class that is the Binary Search Tree.
b. Create a class called `PlayerBST` in the file you just created. Within that class, create the initialiser method and create an instance variable that will contain the root of the search tree. Set that variable to `None`, initially. Create a property for it.
c. Add the file to your local repository without committing yet.
d. Add a new file to the folder `app` called `player_bnode.py`. This will contain the class that holds each node in the BST.
e. Create a class called `PlayerBNode` in the file you just created. Create an initialiser method that accept one argument: `player`. It should assign that value to a private instance variable. Create a property for that value too. Add two more private instance variable: one that points to the left sub-tree and one that points to the right sub-tree. Initialise those variables with `None`. Create getters (or properties) and setters for these values.
f. Add the new file to your local repository.
g. Commit all files and push the latest changes to your remote repository on GitHub.

### Step 5 – Insert Player in Binary Search Tree

In this step, you will create the code to insert `Player` objects into the BST. This step has multiple parts to it:

| | |
|---|---|
| **Qualification national code and title** | ICT50220 Dip Advanced Programming |
| **Unit/s national code/s and title/s** | ICTPRG535 – Build advanced user interfaces<br>ICTPRG547 – Apply advanced programming skills in another language |

a. Add a method called `insert` to the `PlayerBST` class. It should take a `Player` object as the only argument. Please note that it should use the `Player` object's `name` property as the key, since we want to search by name.
b. Implement the function (recursively) so it follows the rules of the BST:
    1. The left subtree of a node contains only nodes with keys less than the node's key
    2. The right subtree of a node contains only nodes with keys greater than the node's key
    3. The left and the right subtree themselves must also be Binary Search Trees
    4. There must not be duplicate nodes (if a node with a key already exists, you *may* update the value/item – which is a `Player` object – of that key.)
c. Create unit tests to test the behaviour of the BST's `insert` method.
d. Commit your changes and push them to your remote repository on GitHub.

## Step 6 – Implement search functionality

In this step, you will implement the search functionality. This step has multiple parts to it:

a. Add a method `search` that accepts a single argument: `name`. Remember that the name is the key for each node in the BST.
b. Implement the following algorithm (recursively):
    1. If the root is `None` or the root node's key matches the searched name, return it.
    2. If the key is less than the root's key, search the left subtree.
    3. If the key is greater than the root's key, search the right subtree.
c. Create unit tests to test the behaviour of the BST's `search` method.
d. Commit your changes and push them to your remote repository on GitHub.

## Step 7 – Optimise Binary Search Tree

In the final step, you will optimise the BST by creating a *Balanced* BST. This step has multiple parts to it:

a. Create a sorted list from the current (non-balanced) BST.
b. Pick the middle element and make that the root of the new Balanced BST.
c. Now, perform the following steps until you run out of elements. *Do this recursively.*
    1. Get the middle of the left and make it the left child of the root from step b.
    2. Get the middle of the right and make it the right child of the root from step b.
d. Commit your changes and push them to your remote repository on GitHub.

| | |
|---|---|
| **Qualification national code and title** | ICT50220 Dip Advanced Programming |
| **Unit/s national code/s and title/s** | ICTPRG535 – Build advanced user interfaces<br>ICTPRG547 – Apply advanced programming skills in another language |

## Step 8 – Knowledge Question

With the newly balanced BST, how many steps does it take *at most* to find an existing item in the search tree?

## Submitting your work

ZIP your entire project into a single file.

Make sure to *remove the Python Virtual Environment folder (called* `venv` *or* `.venv`*) from the ZIP-file* before uploading it into the Blackboard assessments area.