Government of Western Australia
North Metropolitan TAFE

Portfolio Assessment Task 1

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

Assessment type (☑):

☐ Questioning (Oral/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/or 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 5, 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 1

Qualification national code and title	ICT50220 Dip Advanced Programming
Unit/s national code/s and title/s	ICTPRG535 – Build advanced user interfaces 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 implementing a dynamic data structure (a Double Linked List, to be precise). It is expected that the software you create is of a high quality and passes a certain level of testing. You will be provided with guidelines regarding the extent of testing you must perform.

It is expected that the software you create is of a high quality and passes a certain level of testing. You will be provided with guidelines regarding the extent of testing expected of you.

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 as SRU has adopted it as their code style.

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. Please note that you'll need the results of the tasks outlined in this assessment tool for future tasks as well.

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 1

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

Assessment Instrument:

Step 1 – Set up local and remote git repositories

In the first step, we set up the project, create a local Git repository and remote Git repository on GitHub. This step has multiple parts to it:

a. Create an empty project. You can this from the command line or you can use your preferred IDE like PyCharm to do this for you. Name the project `SRUS-XXX-Games`, where `XXX` should be replaced with your initials. If your name is Alex Johnson, the project should be called `SRUS-AJ-Games`.
b. Add a file called `.gitignore` to your project. Ensure that it contains useful entries for your project (Python, PyCharm, etc.)
c. Create two folders within your project: `app` and `test`. Within each of those, create an empty file called `.keep`. This will ensure that the folders will be added to your local repository in the next step. You may remove these files once you've added Python source files to the folders.
d. Initialise the project folder as a local Git repository. From the command line, run `git init .` Alternatively, you can use the IDE for this purpose. Commit your local changes with a commit message that reads something like "Initial commit."
e. Create a repository on GitHub. Do *not* have GitHub create additional files like README, LICENSE, etc. Next, set up that repository as the remote for your local repository by following the provided instructions.
f. Push the local changes to your remote repository.

Step 2 – Create core class and unit tests

In the second step, we'll create the core class for this task and a few unit tests. This step has multiple parts to it:

a. Within the `app` folder, create a file `player.py`. This will hold the Player class that you will create in the next steps. Immediately *add* the file to your local repository. You do not have to *commit* the file yet.
b. Within the file you just made, create a new class called `Player`. Add an initialiser (constructor) function `__init__` that takes two arguments: a unique id (a string) and a player name (also a string). Have the initialiser create two *private* instance variables and assign the provided values.
c. Create two properties for this class, one called `uid` and one called `name`. Have them return the values of the corresponding instance variables.

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

d. Add a method `__str__` to the `Player` class, which returns a human readable string representing the player object.
e. Within the folder `test`, create a file called `player_test.py`. Add to your local repository without committing it yet.
f. Write 2 unit tests that test the behaviour of the class based on its current functionality (i.e., test the properties `uid` and `name`).
g. Commit all changes, then push them to your remote repository.

Step 3 – Prepare the Double-Linked List implementation

In this step, you'll be creating the class that will implement the double linked list. This step has multiple parts to it:

a. Create a new Python source file called `player_list.py` within the `app` folder and add the file to your repository (no need to commit yet). Within this file, you will create a new class that implements a double linked list (sometimes called a *doubly* linked list), so the game that uses the list can easily move from player to player in both directions, which is one of the requirements.
b. Within the file, create a new class called `PlayerList`. Add an initialiser method and within that, create a single *private* instance variable that will point to the head of the list. Initialise that instance variable to `None`.
c. Commit the latest changes to your repository.

Step 4 – Create the Player class

In this step, you'll be adding a new class that will hold the player object and acts as a node in the linked list. This step has multiple parts to it:

a. Create a new Python source file called `player_node.py` within the `app` folder. Add it to your repository without committing it.
b. Within that file, create a new class called `PlayerNode`. Add an initialiser method that takes a single argument called `player` and create 3 *private* instances variables. Assign the variable `player` to a private instance variable. The other two instance variables will point to the next and the previous node. Initialise those instance variables to `None`.
c. Create the necessary getters (properties) and setters for this class. You may not need a setter for every instance variable.
d. Create a property called `key`, which simply returns the `uid` property of the instance variable for the player object.
e. Add a method `__str__` to the `PlayerNode` class, which returns a string representing the node object.
f. Commit the latest changes to your repository and push them to the remote repository.

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

Step 5 – Implement the Double-Linked List

In this step, you will flesh out the implementation of the double linked list called `PlayerList`. This step has multiple parts to it:

a. Create a method that allows you to insert a new node at the head of the list. You should consider two possibilities: the list is empty, or the list is not empty. For this purpose, you may consider adding a new method or property that tells you whether the list is empty. For example, you might call it `is_empty` and have it return a `Boolean`.
b. Write unit tests for both situations. Created a file called `player_list_test.py` inside the test folder for this purpose. Make sure to add it to your repository.
c. After implementing the code and the tests, run the tests to ensure your code works according to specifications. Commit the latest changes and push them to your remote repository.

Step 6 – Add code optimisation

In this step, you'll do a simple code optimisation that will make working with linked lists easier. In many cases, users of a linked list will want to be able to access it from either the head or the tail. For this purpose, we'll be adding a new instance variable to point to the tail. This step has multiple parts to it.

a. In the initialiser, add a private instance variable that will point to the tail of the list. Initialise it to `None`. Consider whether you'll need to implement a property for this value.
b. Update the code you created in the previous step to update the newly created instance variable when you insert an item. Make sure to either update the tests if necessary or create new tests.
c. Commit your changes and push them to your remote repository on GitHub.

Step 7 – Add functionality to Double-Linked List

In this step, you'll be adding some more functionality to the linked list. This step has multiple parts to it:

a. Add a method to insert an item at the tail of the linked list. You need to consider (again) that your list may or may not be empty at this point.
b. Add a unit test that covers this use case.
c. Commit your changes.
d. Add two methods: one to delete an item from the head of the list and one to delete an item from the tail of a list.
e. Add unit tests that cover these use cases.
f. Commit your changes.
g. Add a method to delete an item from the linked list based on its key.
h. Add a unit test that covers this use case.
i. Commit your changes and push all the latest changes 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 ICTPRG547 – Apply advanced programming skills in another language

Step 8 – Show entire list

In the final step of this assessment, you'll add functionality to show the entire list. This step has multiple parts to it:

a. Add a method called `display` to the `PlayerList` class. It should accept one default argument called `forward` (or similar) with the default value being `True`.
b. Implement code that iterates over the linked list either head to tail (`forward=True`) or tail to head (`forward=False`). Print each node in a way that is sensible for human readers.
c. Commit the changes to your local repository and finally push all changes to your remote repository on GitHub.

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.