MODULE NAME:	MODULE CODE:
PROGRAMMING 3B	PROG7312

ASSESSMENT TYPE: POE (PAPER ONLY)

TOTAL MARK ALLOCATION: 100 MARKS

TOTAL HOURS: A MINIMUM OF 45 HOURS IS SUGGESTED TO COMPLETE THIS ASSESSMENT

By submitting this Portfolio of Evidence (POE), you acknowledge that you have read and understood all the rules as per the terms in the registration contract, in particular the assignment and assessment rules in The IIE Assessment Strategy and Policy (IIE009), the intellectual integrity and plagiarism rules in the Intellectual Integrity Policy (IIE023), as well as any rules and regulations published in the student portal.

INSTRUCTIONS:

1. ***No material may be copied from original sources, even if referenced correctly, unless it is a direct quote indicated with quotation marks. No more than 10% of the assignment may consist of direct quotes.***
2. ***Make a copy of your POE before handing it in.***
3. *POE must be typed unless otherwise specified.*
4. *All work must be adequately and correctly referenced.*
5. *Begin each section on a new page.*
6. *Follow all instructions on the assignment cover sheet.*
7. *This is an individual POE.*

Referencing Rubric

Providing evidence based on valid and referenced academic sources is a fundamental educational principle and the cornerstone of high-quality academic work. Hence, The IIE considers it essential to develop the referencing skills of our students in our commitment to achieve high academic standards. Part of achieving these high standards is referencing in a way that is consistent, technically correct and congruent. This is not plagiarism, which is handled differently.

Poor quality formatting in your referencing will result in a penalty **of a maximum of ten percent being deducted from the percentage awarded**, according to the following guidelines. Please note, however, that **evidence of plagiarism in the form of copied or uncited work (not referenced), absent reference lists, or exceptionally poor referencing, may result in action being taken in accordance with The IIE's Intellectual Integrity Policy (0023)**.

Markers are required to provide feedback to students by indicating **(circling/underlining) the information that best describes the student's work.**

Minor technical referencing errors: 5% deduction from the overall percentage – the student's work contains **five or more errors** listed in the minor errors column in the table below.

Major technical referencing errors: 10% deduction from the overall percentage – the student's work contains **five or more errors** listed in the major errors column in the table below**.**

If both minor and major errors are indicated, then 10% only (and not 5% or 15%) is deducted from the overall percentage. The examples provided below are not exhaustive but are provided to illustrate the error.

Required: **Technically correct referencing style**	**Minor errors in technical correctness of referencing style** **Deduct 5% from percentage awarded**	**Major errors in technical correctness of referencing style** **Deduct 10% from percentage awarded**
Consistency • The same referencing format has been used for all in-text references and in the bibliography/reference list.	Minor inconsistencies. • The referencing style is generally consistent, but there are one or two changes in the format of in-text referencing and/or in the bibliography. • For example, page numbers for direct quotes (in-text) have been provided for one source, but not in another instance. Two book chapters (bibliography) have been referenced in the bibliography in two different formats.	Major inconsistencies. • Poor and inconsistent referencing style used in-text and/or in the bibliography/ reference list. • Multiple formats for the same type of referencing have been used. • For example, the format for direct quotes (in-text) and/or book chapters (bibliography/ reference list) is different across multiple instances.
Technical correctness • Referencing format is technically correct throughout the submission. • Position of the reference: a reference is directly associated with every concept or idea. • For example, quotation marks, page numbers, years, etc. are applied correctly, sources in the bibliography/reference list are correctly presented.	**Generally, technically correct with some minor errors.** • The correct referencing format has been consistently used, but there are one or two errors. • Concepts and ideas are typically referenced, but a reference is missing from one small section of the work. • Position of the references: references are only given at the beginning or end of every paragraph. • For example, the student has incorrectly presented direct quotes (in-text) and/or book chapters (bibliography/reference list).	**Technically incorrect.** • The referencing format is incorrect. • Concepts and ideas are typically referenced, but a reference is missing from small sections of the work. • Position of the references: references are only given at the beginning or end of large sections of work. • For example, incorrect author information is provided, no year of publication is provided, quotation marks and/or page numbers for direct quotes missing, page numbers are provided for paraphrased material, the incorrect punctuation is used (in-text); the bibliography/reference list is not in alphabetical order, the incorrect format for a book chapter/journal article is used, information is missing e.g. no place of publication had been provided (bibliography); repeated sources on the reference list.
Congruence between in-text referencing and bibliography/ reference list • All sources are accurately reflected and are all accurately included in the bibliography/ reference list.	**Generally, congruence between the in-text referencing and the bibliography/ reference list with one or two errors.** • There is largely a match between the sources presented in-text and the bibliography. • For example, a source appears in the text, but not in the bibliography/ reference list or vice versa.	**A lack of congruence between the in-text referencing and the bibliography.** • No relationship/several incongruencies between the in-text referencing and the bibliography/reference list. • For example, sources are included in-text, but not in the bibliography and vice versa, a link, rather than the actual reference is provided in the bibliography.
In summary: the recording of references is accurate and complete.	In summary, at least **80%** of the sources are correctly reflected and included in a reference list.	In summary, at least **60%** of the sources are incorrectly reflected and/or not included in reference list.

Overall feedback about the consistency, technical correctness and congruence between in-text referencing and bibliography:

.....

.....

Portfolio of Evidence (PoE) — Background

Your **local library** has asked you to develop a **software application** to **train** library users and novice librarians in the use of the **Dewey Decimal Classification** system.

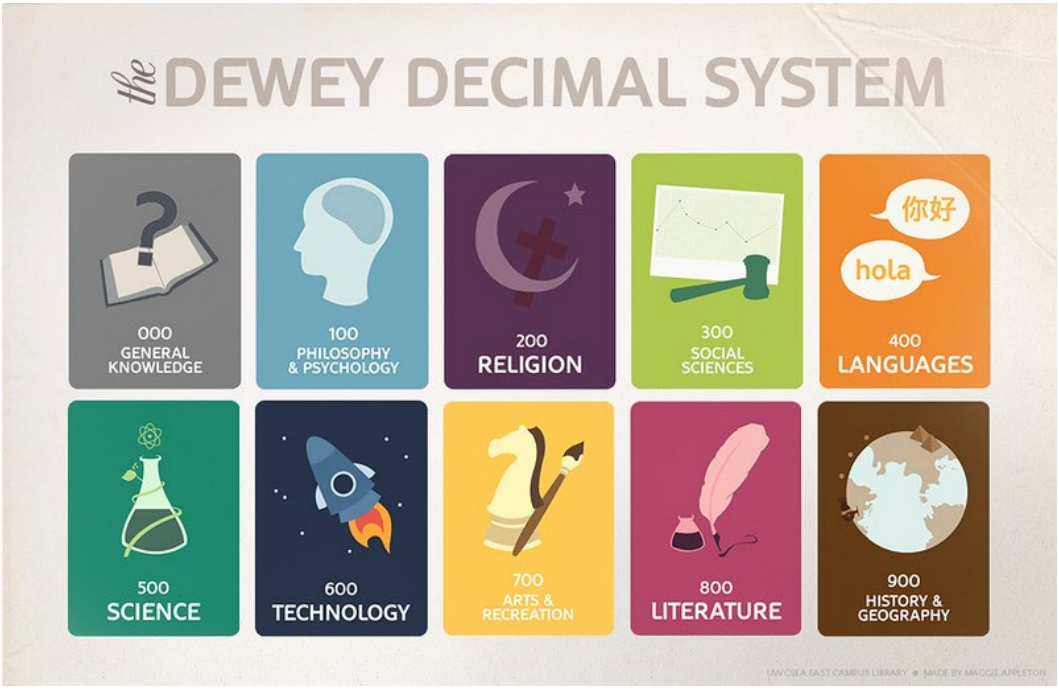

Figure 1: The Dewey Decimal System (Gowing, 2010)

> "The **Dewey Decimal System** organizes information into 10 broad areas, which are broken into smaller and smaller topics. Different topics are assigned numbers, known as 'call numbers'. For example, 'Tigers' are given the number 599.756. To see what books the library currently has in on tigers, go to the nonfiction shelves and find the books that have that number on their spine label." (Mcpl.info, 2020)

The Dewey Decimal System is fundamental to how everything is organised in the library. But the head librarian has found that even novice librarians sometimes get **bored** when they must learn about the details of the system. The purpose of the training software that you will develop for the library, is to make **learning** about the system **fun** and **engaging**.

After using the training software, the user must be able to:

- Identify which broad **area** a book belongs to.
- Find the **call number** for a **specific topic**.
- Correctly **replace** a book in its position on a shelf.

You can use whatever user interface technology you think is most suitable to solve this problem. However, the library does require that the application must be written in C# and must be buildable using Visual Studio 2019.

Bibliography

- Gowing, S. (2010). *Dewey Decimal System Poster*. [Online]. Flickr. Available at: https://www.flickr.com/photos/appletonmaggie/5907672591 [Accessed 14 July 2022].
- Mcpl.info. (2020). *How To Use the Dewey Decimal System*. [Online]. Available at: https://mcpl.info/childrens/how-use-dewey-decimal-system [Accessed 14 July 2022].

Instructions

Complete the parts below to provide the required software. A list of items to be **submitted** for **each part** is specified – make sure you **submit everything** that is required!

Part 1 — Replacing Books on their Shelves (Marks: 100)

Learning Units: LU1 – LU2

This part 1 has two components – **Research** (20 marks) and **Implementation** (80 marks).

Part 1: RESEARCH

The head librarian wants the software application to be **fun** to use. So, do **online research** about **gamification features** and **choose** a **feature** to implement in your application. Here is an article to get you started with your research:

Krasko, A. (2018). *5 Most Popular Gamification Features (With Examples)*. [Online]. eLearning Industry. Available at: https://elearningindustry.com/gamification-features-5-most-popular-examples [Accessed 14 July 2022].

In a Word document:

- **List five gamification features** you **considered** during your research.
- Explain in **200 to 300 words** which **gamification feature** you chose to implement and **why**.
- Remember to **reference** the sources that you used!

Note: If your explanation exceeds 300 words, any work after this point will not be marked.

Part 2: IMPLEMENTATION

Finding the correct place for a book on the shelves in the library requires librarians to be able to **sort call numbers** in **numerical** and then **alphabetical** order. For example, the **call number** for the prescribed book for this module is **005.73 JAM** – the numbers indicate the book's topic, and the letters are the first three letters of the author's surname.

Write a C# software application that fulfils the following requirements:

1. On startup, the application shall allow the user to **choose** between three tasks:
    a. Replacing books.
    b. Identifying areas.
    c. Finding call numbers.
2. For this first task, only **Replacing books** will be implemented – disable the other two options for now.
3. When the user selects **Replacing books**, the application shall **randomly generate 10** different **call numbers** and **display** them to the user.
4. The application shall allow the user to **reorder** the call numbers in ascending order and **check** whether the user got the **ordering right**.
5. Implement the **gamification feature** that you identified to **motivate users** to keep learning.

Technical requirements:

1. Make use of a **list** to store the **generated call numbers**.
2. Choose an **appropriate sorting algorithm** to sort the call numbers to check the order that the user puts them in.

Create a **readme file** that explains how to compile and run the program. In addition, include any usage instructions that may be necessary for your lecturer to start using the application.

> **Note:** No marks will be awarded for any application functionality if the code does not compile and run.

Submit the following items for this part:

1. A **Word document** containing your **research**.
2. **Source code** for the application.
3. The **readme file** with instructions for how to compile, run and use the software.

Important! You will build on this application in Part 2 and the POE. So, keep a copy of your code in a safe place!

Part 2 — Identifying Areas (Marks: 100)

Learning Units: LU1 – LU4

In this part, you will build on the application you wrote in Part 1. This part has only an **implementation** component.

The head librarian wants everybody to know the top-level categories of the Dewey Decimal System. This is the first step towards finding a book in a library. And in a small library, that might be enough to locate the book you are looking for. Create a match-the-columns question system to allow users to practice the categories.

Here is an example of a match-the-column question to get you started:

Match the definitions in the right-hand column with the terms in the left column:

1	Performance		**A**	A common way of doing things that will not work correctly.
2	Anti-Pattern		**B**	The protection of computing systems and the data that they store or access.
3	Architecture Pattern		**C**	How fast a software application must be able to complete a specific task given a certain system load.
			D	The blending of tasks performed by a company's application development and systems operations teams.
			E	Solutions to common problems at an application-wide level.

Here is a page with good guidelines for creating match-the-column questions:

ClassMaker. (n.d.) *Matching Questions*. [Online]. Available at: https://www.classmarker.com/learn/question-types/matching-questions/ [Accessed 14 July 2022].

The following requirements must be **added** to the **application created in Part 1**:

1. Enable the **Identifying areas** task.
2. When the user chooses the **Identifying areas** task, they should be presented with a user interface where they will **match two columns**: **call number** (top-level only) and **description**.
3. The user shall be allowed to answer **as many questions* as they want to**.
4. The **questions** should **alternate** between matching **descriptions to call numbers** and **call numbers to descriptions**.
5. Each question should have **four randomly selected items** in the **first column** and **seven possible answers** (three of which are incorrect) in the **second column**.
6. Implement a **gamification feature** to **motivate** users to keep using the application. You may use the **same one** as before **or** choose to implement a **different one**.

*** Note:** A **question** in this context is defined as the **whole matching the columns set**, including **both columns**.

Technical requirements:

1. Store the **call numbers** and their **descriptions** in a **dictionary**.

Update the **readme file** that explains how to compile, run and use the program with any new instructions required for this version.

> **Note:** No marks will be awarded for any application functionality if the code does not compile and run.

Submit the following items for this part:

1. **Source code** for the application.
2. The **readme file** with instructions for how to compile, run and use the software.

Important! You will build on this application in the POE. So, keep a copy of your code in a safe place!

POE — Finding Call Numbers (Marks: 100)

Learning Units: LU1 – LU5

In this part, you will build on the application you wrote in Part 1 and 2.

Important: Remember to incorporate any **feedback** provided by your **lecturer** on Parts 1 and 2 before implementing the new functionality. Marks will be awarded for this (see rubric). Also, include a file that lists the **changes** you have made.

This part has two components – **Research** (20 marks) and **Implementation** (80 marks).

Part 1: RESEARCH

Do **online research** to find **call numbers** and their **descriptions** to the **nearest integer** (i.e., no need to get into the digits after the decimal point).

In a Word document, create a **multi-level list** showing the call numbers, for example:

- 700 Arts & Recreation
  - 750 Paining
    - 751 Techniques, procedures, apparatus, equipment, materials, forms
    - 752 Color

Example from: http://library.mysek.school/index.php?lvl=indexint_see&id=5676 [Accessed 14 July 2022].

The list must contain **at least 100 unique entries** across all three levels. Remember to **reference** your sources!

Part 2: IMPLEMENTATION

There are far too many categories in the Dewey Decimal system for any human to remember them all. But it is essential for users to understand how to find something in the hierarchy. So, create a quiz where the user drills deeper and deeper into the hierarchy until they find the correct answer.

The following requirements must be **added** to the **application created in Part 2**:

1. Create a **file** containing the **data** that was gathered in the **research** part of this task in a format that your application can read.
2. Enable the **Finding call numbers** task.
3. When the user chooses **Finding call numbers**, the application must **load** the Dewey Decimal classification data into **memory** from the **file** created in Step 1.
4. The quiz must work as follows:
    a. For each question, **randomly** select a **third-level entry** from the data, for example, **752 Color**. Display only the **description**, **not** the **call number**.
    b. Display **four top-level options** to the user to **choose** between, one of which must be the **correct one** and the other three **randomly selected incorrect** answers. For example:
    000 General
    400 Language
    700 Arts & Recreation (Correct answer)
    800 Literature
    c. For the **options**, display **both** the **call number** and **description**. Display the options in **numerical order** by call number.
    d. If the user selects the **correct option**, show them **four options** from the **next level** until the most detailed level is reached.
    e. If the user selects the **wrong option** anywhere along the way, indicate this and then ask the next question.
5. Implement a **gamification feature** to **motivate** users to keep using the application. You may use the **same one** as before **or** choose to implement a **different one**.

Technical requirements:

1. Make use of a **tree** to store the **data** in memory.

Update the **readme file** that explains how to compile, run and use the program with any new instructions required for this version.

Note: No marks will be awarded for any application functionality if the code does not compile and run.

Submit the following items for this part:

1. A **Word document** containing the **multi-level list**.
2. **The application's source code** must include the **Dewey Decimal data file**.
3. The **readme file** with instructions for how to compile, run and use the software.
4. A file listing the **updates** you have made based on **feedback** from your lecturer.

Appendix A

Assessment Sheet (Marking Rubric)

Please note: Tear off this section and **attach** it to your work when submitting it.

MODULE NAME:	**MODULE CODE:**
PROGRAMMING 3B	**PROG7312**

STUDENT NAME:
STUDENT NUMBER:

RUBRIC 1 — SKELETON OUTLINE	**Levels of Achievement**				**Feedback**
In order to be awarded full marks for these elements of Part 1, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
Research: List of five gamification features.	**5** Five distinct gamification features are listed.	**3—4** Three or four distinct gamification features are listed.	**1—2** One or two distinct gamification features are listed.	**0** No gamification features are listed.	
Research: Introduction and conclusion.	**5** An excellent introduction and	**3—4** A good introduction and	**1—2** Either the introduction or	**0** No introduction or conclusion is	

RUBRIC 1 — SKELETON OUTLINE	Levels of Achievement				Feedback
In order to be awarded full marks for these elements of Part 1, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
	conclusion that adds value to the discussion are included.	conclusion are included but do not link perfectly with the rest of the discussion.	conclusion is missing, or both are entirely unrelated to the rest of the discussion.	included in the discussion.	
Research: Motivation for choice of gamification feature.	**9—10** An excellent motivation is provided that includes the benefits of the specific feature and motivates why it was chosen.	**7—8** A good level of detail is included in the motivation with only one or two things that could be improved.	**4—6** Some details are provided in the discussion, but the motivation is not presented completely logically.	**0—3** No motivation is provided, or the motivation is entirely illogical.	
App Functionality: Allow the user to choose which task to practice.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	

RUBRIC 1 — SKELETON OUTLINE	**Levels of Achievement**				**Feedback**
In order to be awarded full marks for these elements of Part 1, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
App Functionality: Display ten randomly generated call numbers.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	
App Functionality: The user can change the order of the call numbers.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	
App Functionality: The app checks whether the user got the ordering correct.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	
App Functionality: A gamification feature is implemented.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	

RUBRIC 1 — SKELETON OUTLINE	**Levels of Achievement**				**Feedback**
In order to be awarded full marks for these elements of Part 1, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
App Logic: A list is used to store the call numbers.	**5** A list is consistently used throughout the application to store the call numbers.	**3—4** A list is used in most places in the app to store the call numbers.	**1—2** A list is used only in some places with arrays or a different data structure being used in others.	**0** A list is not used at all to store the call numbers.	
App Logic: An appropriate sorting algorithm is used to sort the call numbers.	**9—10** The sorting algorithm works perfectly.	**7—8** The sorting algorithm works with only one or two issues.	**4—6** The sorting algorithm works some of the time, but there are lots of issues.	**0—3** The call numbers are not sorted, or the algorithm does not work for call numbers.	
Coding Standards: The code is well structured and documented.	**5** The code is excellently structured, easy to read, and with sufficient detail in the comments.	**3—4** The code structure can be somewhat improved or too few comments included.	**1—2** The code is not well structured but somewhat readable, and very few comments are included.	**0** The code is poorly structured, no naming convention used and no comments included.	

RUBRIC 1 — SKELETON OUTLINE	**Levels of Achievement**				**Feedback**
In order to be awarded full marks for these elements of Part 1, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
Documentation: The readme file provides enough information to run the app.	**5** An excellent readme file is included with all the relevant information included.	**3—4** The readme file contains some information but could be more complete.	**1—2** The readme file contains very little useful information.	**0** No readme file was submitted.	
Other marks: The app is easy to use.	**5** The app is straightforward and intuitive to use.	**3—4** The app is relatively easy to use, with only one or two usability issues.	**1—2** The app can use used, but it is hard to know how to access its features.	**0** The app is impossible to use.	
Other Marks: Advanced features not covered in class (Bonus Marks).	**[5]** Extensive use of advanced features.	**[3—4]** Good use of advanced features.	**[1—2]** Limited use of advanced features.	**[0]** No advanced features were used.	
Part 1 TOTAL					**/100**

RUBRIC 2 — SKELETON OUTLINE	**Levels of Achievement**				**Feedback**
In order to be awarded full marks for these elements of Part 2, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
App Functionality: The user can choose the new Identifying areas task.	**5** The feature works perfectly without any errors.	**3—4** The feature is well implemented with only one or two bugs.	**1—2** The feature is implemented, but there are lots of bugs.	**0** The feature is not implemented or does not work at all.	
App Functionality: The user is presented with a randomly generated match-the-columns question, with more answers than questions.	**16—20** The feature works perfectly without any errors.	**10—15** The feature is well implemented with only one or two bugs.	**5—9** The feature is implemented, but there are lots of bugs.	**0—4** The feature is not implemented or does not work at all.	
App Functionality: The questions should alternate between descriptions to call numbers and call numbers to descriptions.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	
App Functionality: The user can complete the	**9—10**	**7—8** The feature is well implemented with	**4—6** The feature is implemented, but	**0—3** The feature is not implemented or	

RUBRIC 2 — SKELETON OUTLINE	**Levels of Achievement**				**Feedback**
In order to be awarded full marks for these elements of Part 2, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
match the columns question.	The feature works perfectly without any errors.	only one or two bugs.	there are lots of bugs.	does not work at all.	
App Functionality: The app checks whether the selected answers are correct.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	
App Functionality: The app allows the user to keep practising.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	
App Functionality: Gamification feature implemented.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	
App Logic: Data stored in a dictionary.	**9—10**	**7—8**	**4—6**	**0—3**	

RUBRIC 2 — SKELETON OUTLINE	Levels of Achievement				Feedback
In order to be awarded full marks for these elements of Part 2, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
	A dictionary is consistently used throughout to store the data.	A dictionary is used in most places to store the data.	A dictionary is used only in some places to store the data.	A dictionary is not used to store data or is not working at all.	
Coding Standards: The code is well structured and documented.	**5** The code is excellently structured, easy to read, and with sufficient detail in the comments.	**3—4** The code structure can be somewhat improved or too few comments included.	**1—2** The code is not well structured but somewhat readable, and very few comments are included.	**0** The code is poorly structured, with no naming convention and no comments included.	
Documentation: The readme file provides enough information to run the app.	**5** An excellent readme file is included with all the relevant information included.	**3—4** The readme file contains some information but could be more complete.	**1—2** The readme file contains very little useful information.	**0** No readme file was submitted.	
Other marks: The app is easy to use.	**5** The app is straightforward	**3—4** The app is relatively easy to	**1—2** The app can use used, but it is hard	**0** The app is impossible to use.	

RUBRIC 2 — SKELETON OUTLINE	**Levels of Achievement**				**Feedback**
In order to be awarded full marks for these elements of Part 2, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
	and intuitive to use.	use, with only one or two usability issues.	to know how to access its features.		
Other Marks: Advanced features not covered in class (Bonus Marks).	**[5]** Extensive use of advanced features.	**[3—4]** Good use of advanced features.	**[1—2]** Limited use of advanced features.	**[0]** No advanced features were used.	
PART 2 TOTAL					**/100**

RUBRIC 3 (FOR POE) — SKELETON OUTLINE	**Levels of Achievement**				**Feedback**
In order to be awarded full marks for these elements of POE, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
Research: Multi-level list containing at least 100 unique entries and goes to level 3.	**16—20** 100 entries included across all three levels, including a good variety of topics	**10—15** Entries from all three levels were included, but only greater than 50 and less than 100 were included.	**5—9** Only some first and second-level entries include, and less than 50 entries.	**0—4** No list included or only some first-level entries included.	
App Functionality: Replacing books feature (Part 1) working with feedback incorporated.	**5** The feature works perfectly without any errors.	**3—4** The feature is well implemented with only one or two bugs.	**1—2** The feature is implemented, but there are lots of bugs.	**0** The feature is not implemented or does not work at all.	
App Functionality: Identifying areas feature (Part 2) working with feedback incorporated.	**5** The feature works perfectly without any errors.	**3—4** The feature is well implemented with only one or two bugs.	**1—2** The feature is implemented, but there are lots of bugs.	**0** The feature is not implemented or does not work at all.	
Data: File created containing the data to be read by the application.	**5** Complete data file included with all	**3—4**	**1—2**	**0** No data file is included, or the	

RUBRIC 3 (FOR POE) — SKELETON OUTLINE	**Levels of Achievement**				**Feedback**
In order to be awarded full marks for these elements of POE, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
	the data from the research task.	Data file included, but some of the data is missing.	Data file included with very little data.	application cannot read the data.	
App Functionality: Loading data from the file.	**5** The feature works perfectly without any errors.	**3—4** The feature is well implemented with only one or two bugs.	**1—2** The feature is implemented, but there are lots of bugs.	**0** The feature is not implemented or does not work at all.	
App Functionality: The quiz allows the user to select a top-level item and correctly verify the choice.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	
App Functionality: Quiz correctly handles a correct response by showing users options from the next level down.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	

RUBRIC 3 (FOR POE) — SKELETON OUTLINE	**Levels of Achievement**				**Feedback**
In order to be awarded full marks for these elements of POE, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
App Functionality: Quiz correctly handles incorrect answers.	**5** The feature works perfectly without any errors.	**3—4** The feature is well implemented with only one or two bugs.	**1—2** The feature is implemented, but there are lots of bugs.	**0** The feature is not implemented or does not work at all.	
App Functionality: Gamification feature implemented.	**9—10** The feature works perfectly without any errors.	**7—8** The feature is well implemented with only one or two bugs.	**4—6** The feature is implemented, but there are lots of bugs.	**0—3** The feature is not implemented or does not work at all.	
App Logic: Storing data in a tree.	**9—10** A tree is consistently used.	**7—8** A tree is used in most places.	**4—6** A tree is used only in some places.	**0—3** No tree is used or not working.	
Coding Standards: The code is well structured and documented.	**5** The code is excellently structured, easy to read, and with sufficient detail in the comments.	**3—4** The code structure can be somewhat improved or too few comments included.	**1—2** The code is not well structured but somewhat readable, and very few comments are included.	**0** The code is poorly structured, with no naming convention and no comments included.	

RUBRIC 3 (FOR POE) — SKELETON OUTLINE	**Levels of Achievement**				**Feedback**
In order to be awarded full marks for these elements of POE, students need to have:	**Excellent**	**Good**	**Developing**	**Poor**	
	Score Ranges Per Level (½ marks possible)				
Documentation: The readme file provides enough information to run the app.	**5** An excellent readme file is included with all the relevant information included.	**3—4** The readme file contains some information but could be more complete.	**1—2** The readme file contains very little useful information.	**0** No readme file was submitted.	
Other marks: The app is easy to use.	**5** The app is straightforward and intuitive to use.	**3—4** The app is fairly easy to use, with only one or two usability issues.	**1—2** The app can use used, but it is hard to know how to access its features.	**0** The app is impossible to use.	
Other Marks: Advanced features not covered in class (Bonus Marks).	**[5]** Extensive use of advanced features.	**[3—4]** Good use of advanced features.	**[1—2]** Limited use of advanced features.	**[0]** No advanced features were used.	
PART 3 TOTAL					**/100**