WJEC GCSE Computer Science Unit 3 Report: Automated Readability Index

Luca Huelle - Rougemont School

# Scope of the Problem

## Purpose

The purpose of the application will be to allow teachers to check if their resources and teaching materials are suitable for the reading age of their pupils based on the documents’ calculated reading age produced by the Automated Readability Index (ARI). This is to make sure that the resources provided are indeed suitable and that the pupils can read and understand them. The application will consist of two main parts: the entering and storage of each new texts’ details and the search and the retrieval of previously entered texts.

## Data input, processing and output

The storage part of the program will require the following **input data** from the user to function: the title and intended reading age of the text, any text-related keywords and the actual text content. The program will need to store this information for retrieval later. The program will then need to **calculate and process** the number of characters, words and sentences in the text as well as its reading age (using the ARI). The program should **store** this data along with the text’s details and other information. Following processing, the program should **output** the intended reading age and the newly calculated reading age.

The retrieval part of the program will require a separate set of inputs and outputs. The user will **input** a search query, looking for the text using keywords or the calculated reading age. As **processing**, the program will have to search through a database of previously entered texts to find the one(s) required by the user. Finally, it will **output** the search results (text title, intended reading age, calculated reading age, etc.) in the user interface.

## Solution Objectives

The final goal is to produce an application that…

* allows the user to **input** text details and content
* is able to **calculate** the number of characters, words and sentences in the text content
* is able to **calculate** the reading age of a given text using the ARI and the information above
* is able to **store** this data and **output** it to the user after processing
* enables the user to **search** through previously entered texts using keywords or the calculated reading age
* has a user friendly interface (possibly a GUI) that is intuitive to use

# Design

## User Interface Input Facilities

The user interface should provide input facilities, in the specified format (with the respective data structures/types to be used in the program given in brackets), for the following:

* Text details – Separate text entry fields each labelled with the information required including title (string), intended reading age (entered as string and converted to an integer) and keywords (entered as a string of words separated by commas which will then be split (by commas again) into a python list for storage)
* Text content – One extended entry field will be needed here for the entire text content to be entered by the user (string)
* Searching interface – Two text entry fields: one for keywords (again, entered as a string of words separated by commas which will then be split into a list in the actual program for searching); another for the calculated reading age (also entered as string and converted to an integer for use within the program)

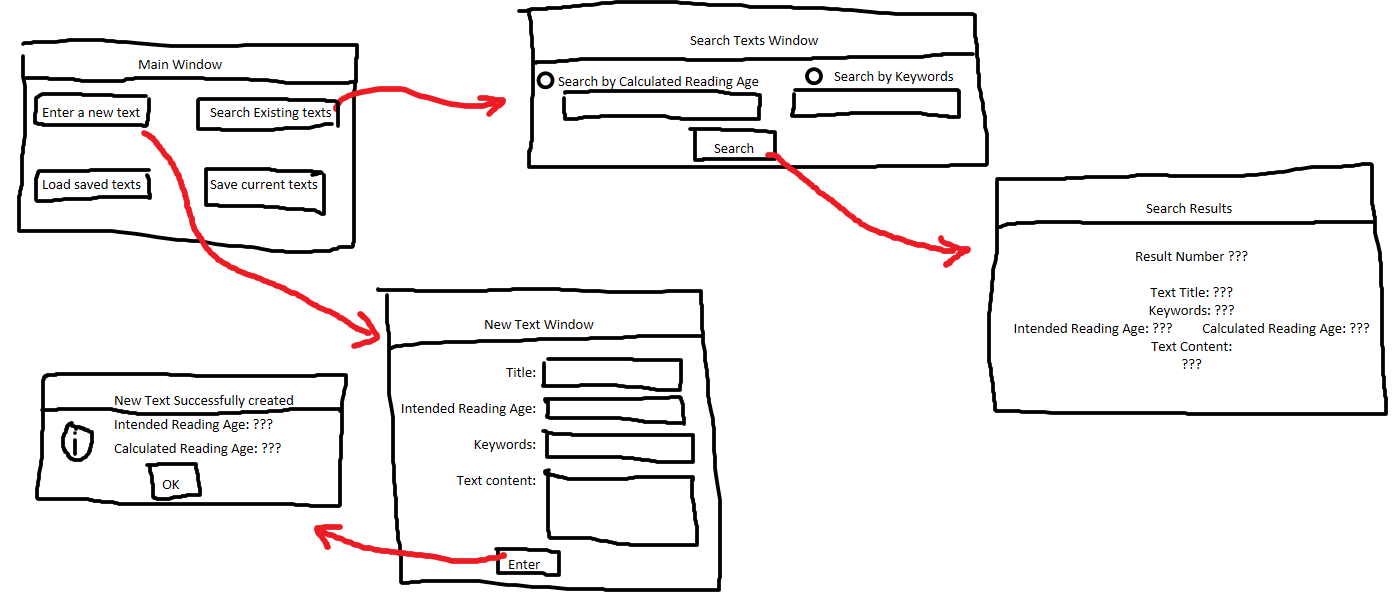
## User Interface Output Facilities

The user interface should provide output facilities for the following:

* The intended reading age and the calculated reading age – both would be displayed as text output by the program, each clearly marked. (If I do create a GUI then this information will be displayed in a separate dialogue box after entering text details and successfully creating a new text.)
* Search results – The results of the search would be displayed as plain text output with individual texts separated by a divider (e.g. ===) and all details of a text clearly marked. (Again, if I do create a GUI then the search results will appear in a different window but in a similar style to below). A condensed version of a proposed text output report for a search is shown below:

Result 1 - Text title: ???  
Keywords: ???  
Intended Reading Age: ??? Calculated Reading Age: ???  
Text Content: ???  
======  
Result 2 - Text title: ???  
…(continued)…

### Possible GUI plan

*(Apologies for the shaky lines – drawn with a computer mouse on screen)*

## Data Structures

To actually store the details of each text I will be using python’s class system as this will allow me to store each text’s details as attributes of a new instance of the ‘Text’ class. To search through the texts, I will then simply compile a list of current texts stored within the system and search through the required attributes. The advantage of using a class based system, instead of a dictionary or 2D list, is that data can be accessed much more easily and intuitively within the program by simply requesting the required *named* attribute of each instance of the ‘Text’ object rather than using a complex sequence of *numerical* indices (e.g. ‘texts[“text1”].name’ instead of ‘texts[“text1”][1]’ which is difficult to read and understand when reading code in retrospect).

### Validation

The program will also need to ensure that all data is entered correctly (i.e. in the correct format, of the right type, etc.) to minimise the risk of errors (e.g. Runtime - division by zero). For the searching aspect of the program, it will check whether all required data has been entered (a presence check), whether the intended reading age entered is an integer (a type check) and whether the keywords are properly separated by commas (a format check) to allow them to be split up correctly by the program. For the text entry side, presence and type checks will be required for the different entry fields as well as a format check, again for the keywords.

### Main Proposed Text Class Data Items/Structures (i.e. attributes) and relevant Validation Methods

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Data Type | Validation Type\* | Validation Details |
| self.title | String | N/A | N/A |
| self.intended\_reading\_age | Integer | Type | is integer |
| self. unformatted\_keywords | String | Length, Format | check if keywords are sensible length and if separated correctly using commas |
| self.keywords | List | N/A | created using validated string above |
| self.calculated\_reading\_age | Integer | N/A | created within program |
| self.content | String | Format | the text content must contain at least one full-stop |

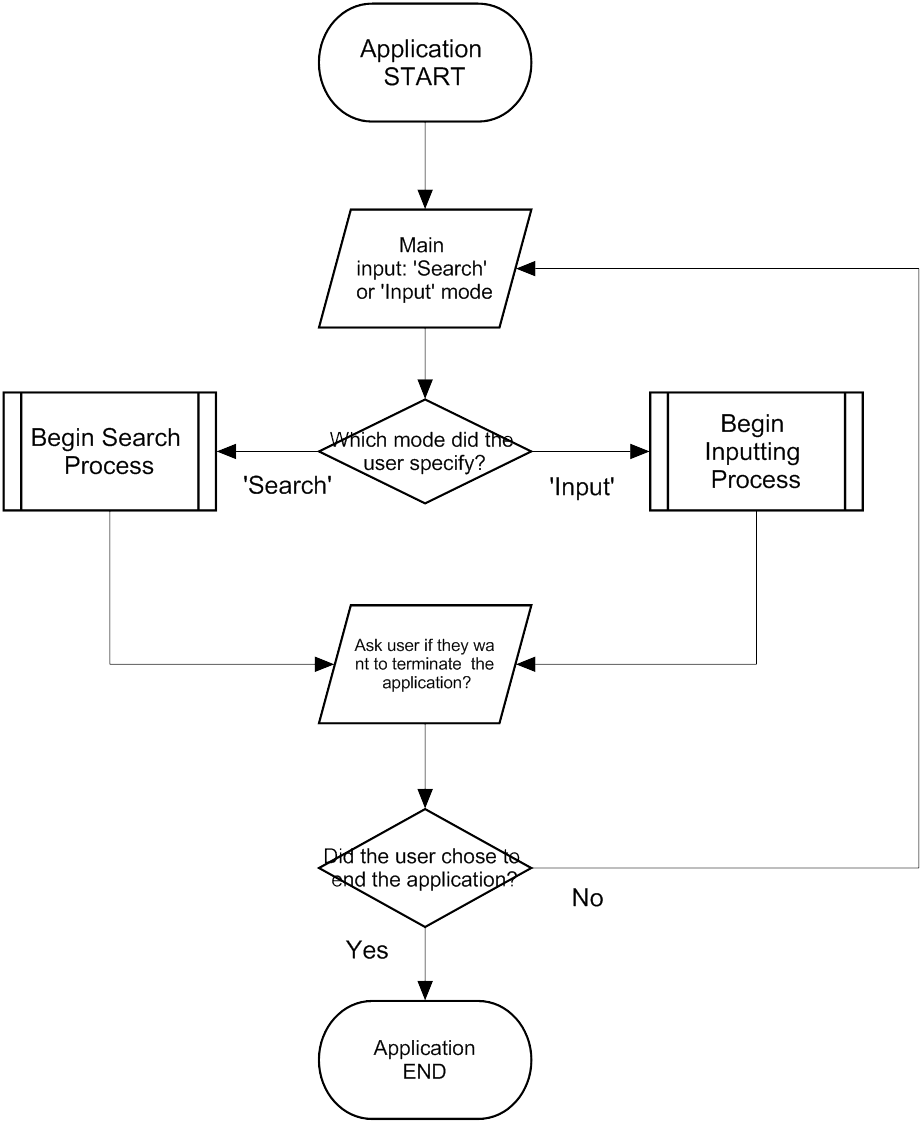
\*in addition to standard presence checks

### The need for authentication routines

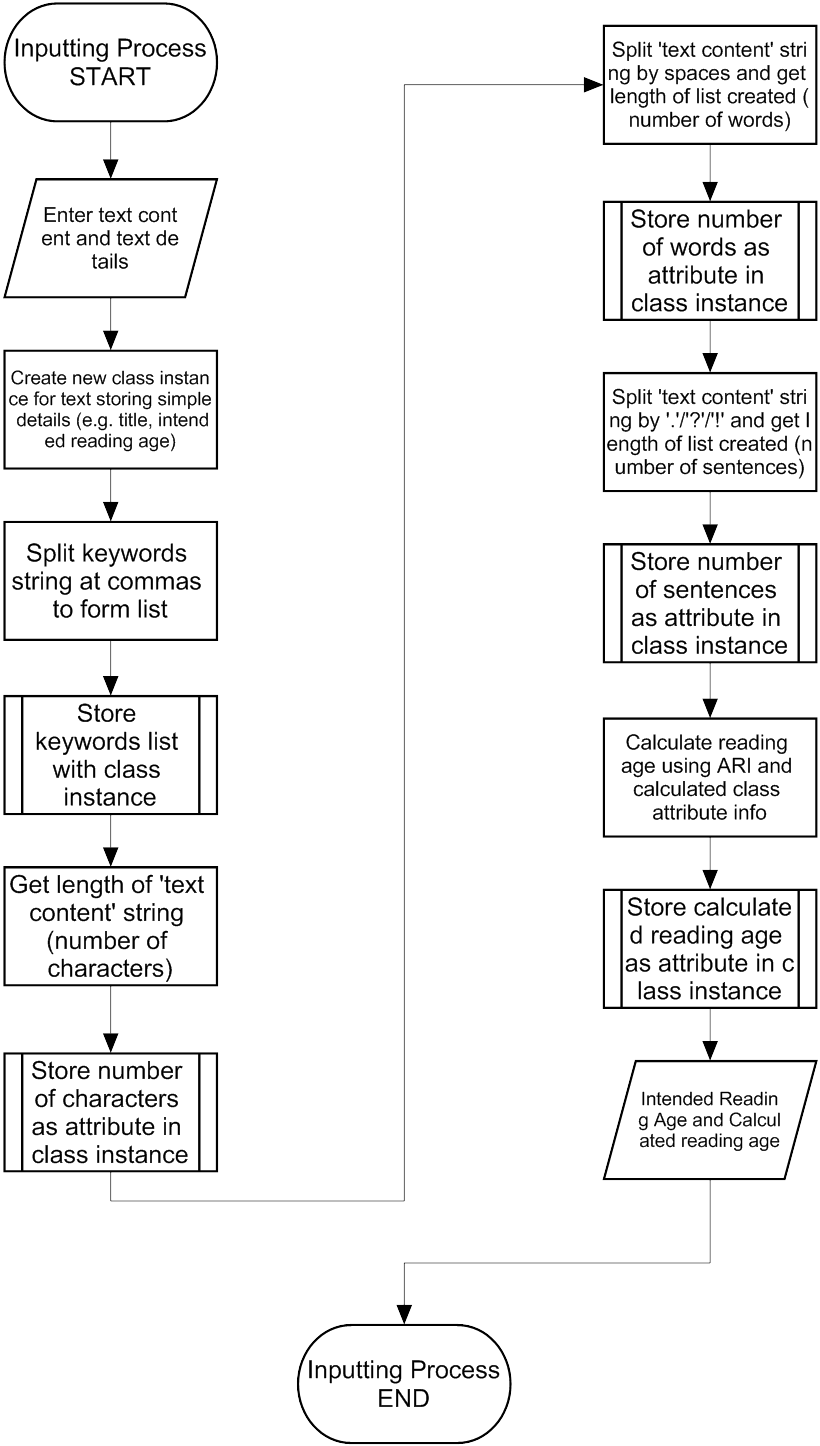
As the program will not be storing any confidential or sensitive information/data there will be no need for a ‘login-like’ system. However, the text data will be stored in a separate file so the possession of this file will act as ‘security’, as only teachers should have access to the file (i.e. only they should have the read/write user access rights to the file and execute rights for the program).

## Processing Stages as Flowcharts

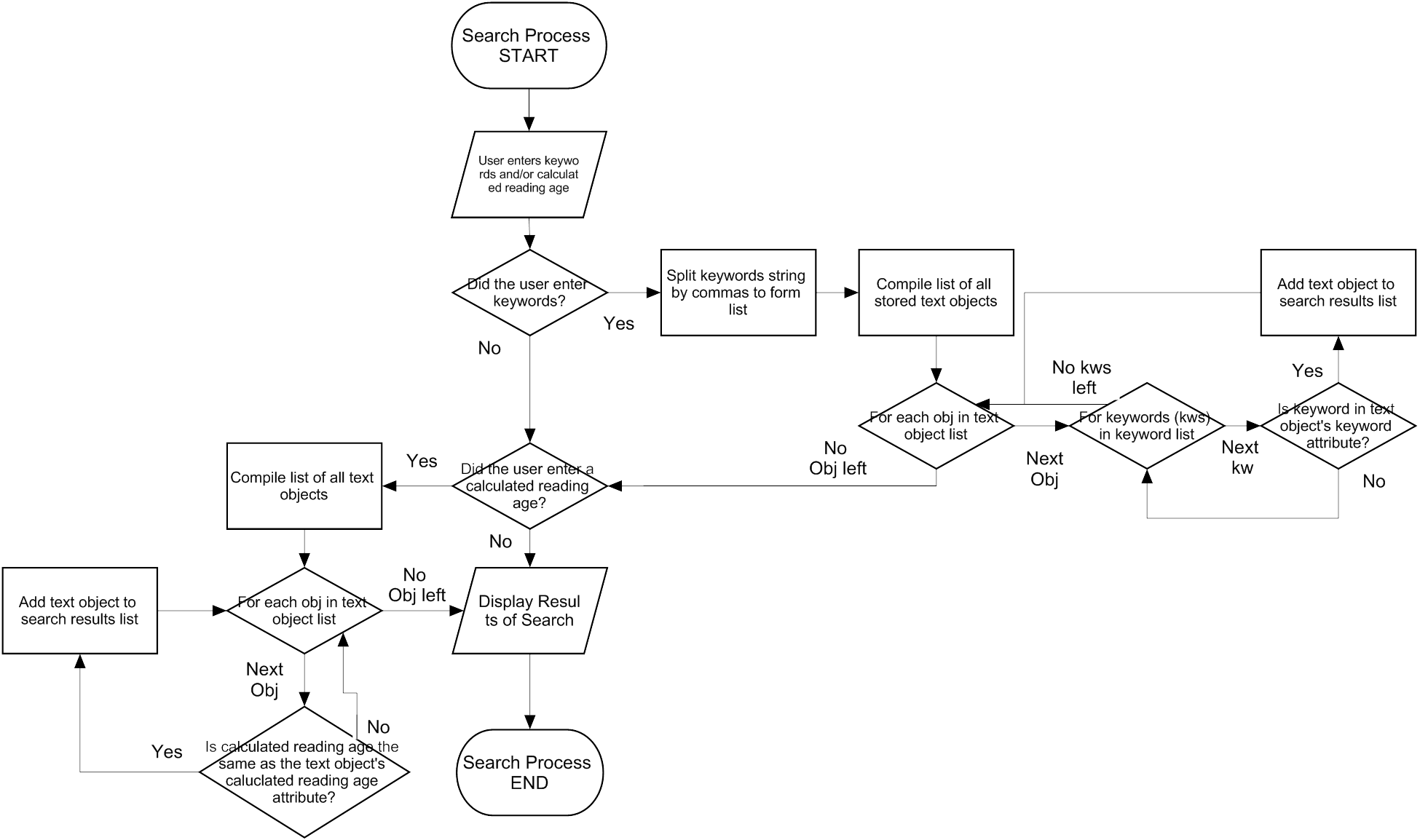
### Main Application



### Calculating reading age using ARI



### Searching Process



# Testing and Test Strategy

## Test Strategy

As I am designing a modular GUI system – with each window’s specific methods (such as searching or creating a new text) isolated to just that window’s class – I can test each class, and its individual methods, on their own. This has the advantage of allowing me to isolate errors in code more easily as they can only be located in a specific line range (i.e. the line range of that class) within the program. It is also very clear from the code (an advantage of python) where functions from outside the current class are called (as they have no ‘self.’ prefix before them), showing me plainly where in the code I need to search for an error. This testing strategy will allow me to test the entire application in parts and then, in turn, as a whole, providing me with an understanding of how the application works and fulfils the success criteria/requirements of the scenario.

## The Purpose of Unit, Integration and Functional Testing

### Unit Testing

Unit testing involves just testing a singular function/method within a class with no access to external databases or resources (such as other functions within the program). This helps to isolate errors further and helps in removing recurring bugs (bugs that occur multiple times throughout the program, e.g. a faulty result is passed through multiple methods) and logic errors (e.g. use of an ‘and’ operator instead of the ‘or’ operator). Unit tests that I conduct will include testing the validate function (i.e. does it perform presence and format checks correctly?) and the creation of a new text using the initialisation method of the Text class.

### Integration Testing

Integration testing focuses more on different class’ (and their methods’) interactions and will allow me to see if the different parts of the program work well work together more effectively and don’t, for instance, share data in the wrong data type. Integration tests I conduct will include testing the method that saves the currently stored texts (this will access the text objects currently stored and then convert them into a .json file) and how the search function tracks through currently stored texts to find those that meet the user’s search criteria.

### Functional Testing

Finally, functional testing will focus on the application behaviour as a completed program. This is not so helpful in finding bugs and errors, as there a simply too many individual pieces of code involved, but does provide a way to measure the final program against the success criteria, allowing for better evaluation.

## Utilising the Results of the Testing Process

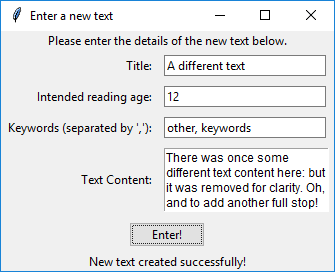
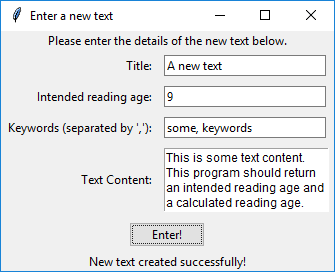
The testing will allow me to verify if the program does indeed meet the success criteria and what areas of the program still require development. For example, I will be able to check if the program does indeed correctly process a text’s data and output the correct calculated reading age as well as if it returns this information in a way which is understandable by, and intuitive to, the user. I will also be able to identify, and then remove, unnecessary pieces of code which may have cluttered the program, causing it to run slower. Finally, the process will also help me to identify parts of the program which are lacking (in user friendliness, UI design, etc.) and could therefore be improved during potential further development.

## Final Evaluative Testing Results

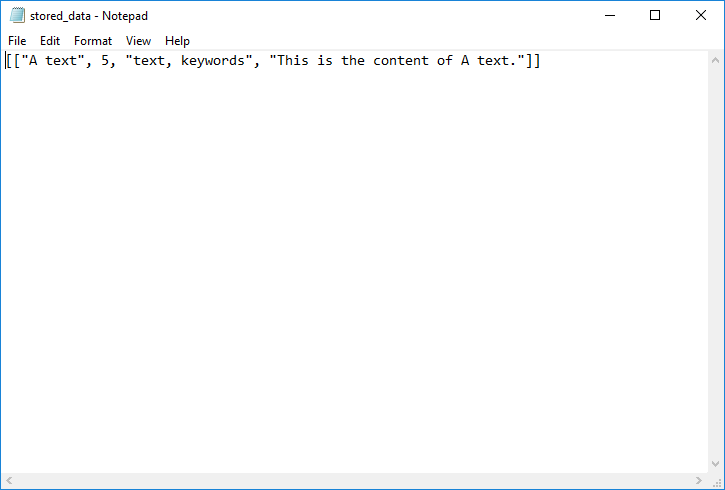
(Test) Type: Unit – U, Integration – I, Functional – F ; ✓ indicates a successful test (i.e. expected and actual output the same); \* indicates testing evidence

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Function/Method | Type | Input Data | Expected Output | Actual Output |  | Comment |
| Text Class – internally handles texts and their data  *\_\_init\_\_ method* | U | “a sentence with no full stop” for content | No full sentences detected | No full sentences detected | ✓ | The program realised there wasn’t a sentence ending punctuation mark in the sentence which would have caused the ARI calculation to raise a division by zero (runtime) error. |
|  | U | “Some realllllllllllllllly long words to increase character count.” for content | Character Count – 65 | Character Count – 65 | ✓ | The program used python’s inbuilt len() function to count the number of characters in the inputted string. |
|  | U | “A sentence with six different words. Another sentence with more!” for content | Word Count – 10 | Word Count – 10 | ✓ | The program correctly counted the words in the string by splitting it up by spaces. |
|  | U | “Hello World!” for content | Calculated Reading Age – 8 | Calculated Reading Age – 7.83 |  | I had forgotten to use the ceiling function to round the result up as required for the ARI. |
|  | U | “a sentence with a full stop.” for content | Calculated Reading Age – 4 \* | Calculated Reading Age – 3 |  | There was a logical error (BIDMAS) in the actual line of the program where the ARI was calculated leading to the wrong value being calculated. |
| *text\_details method* | I | Typical inputs given\*(text 1 of typical inputs) – return title and intended reading age using method | [“A new text”, 9] | [“A new text”, 9] | ✓ | This method of the Text() class successfully returned a list containing the text’s title and age with the correct data types. |
| MainWindow – Main GUI of application  *load\_data method* | U | ‘Load saved texts’ button with no file present to load from | Save file not found | Save file not found | ✓ | The try and except block in the load\_data method acknowledged the FileNotFoundError and correctly displayed an error message to the user. |
| *save\_data method* | U | ‘Save stored texts’ button with texts stored | Stored\_data.json successfully created | Stored\_data.json successfully created \* | ✓ | The program correctly retrieved the required data from all text classes (in the ‘text\_objs’ list) and ‘dumped’ it in a json file. |
| *enter\_new method* | I | ‘Enter a new text’ button | New ‘Enter a new text’ window created | New ‘Enter a new text’ window created | ✓ | A new tkinter TopLevel window was successfully created. |
| *search\_existing method* | I | ‘Search stored texts’ button | New ‘Search stored texts’ window created | NameError |  | I had mistyped the instance declaration statement of the search window - python was unable to find the ‘SerchWindow’ class. |
| InputWindow class – GUI for text creation | U | No input given for any entry fields | Missing Data | Missing Data | ✓ | The program correctly checked all of the entry fields for user input before creating a new text |
|  | U | “nine” for intended reading age field | Non-integer entered | ValueError raised |  | The program was not correctly checking the user’s input to see if it was an integer. I added an additional branch in the validation function (vcmd) to accommodate for this. |
|  | I | “an unsplit keyword” for keywords;  Then ‘no’ when prompted if to continue | No new text created. | No new text created. | ✓ | The validate function successfully returned False after the user selected that they did not want to proceed with the incorrectly formatted inputted keyword list. |
|  | F | Typical inputs given\*(text 1 of typical inputs) | New Text Created Successfully!  Intended Reading Age: 9  Calculated Reading Age: 12 | New Text Created Successfully!  Intended Reading Age: 9  Calculated Reading Age: 12 | ✓ | The program correctly processed the input data, informed the user of the successful creation process and correctly returned both the intended and calculated reading ages of the text as required. |
| SearchWindow class – GUI for searching texts | F | Texts created using typical inputs\*(both)  “9” for Calculated Reading Age field | No Results found | One search result: “A Different text” |  | The program was, incorrectly, comparing the entered value for *calculated* reading age against each text’s *intended* reading age, leading to the incorrect results being produced. |
|  | F | Texts created using typical inputs\*(both)  “10” for Calculated Reading Age field | One search result: “A Different text” | One search result: “A Different text” | ✓ | The program searched its text database using the inputted calculated reading age, successfully found the one matching result and returned it along with its relevant details. |
|  | F | Texts created using typical inputs\*(both)  “keywords” for Keywords field | Two results (i.e. both texts) | Two results (i.e. both texts)\* | ✓ | Both texts stored were found by the program to match the inputted search criteria so it displayed both texts and their details. |

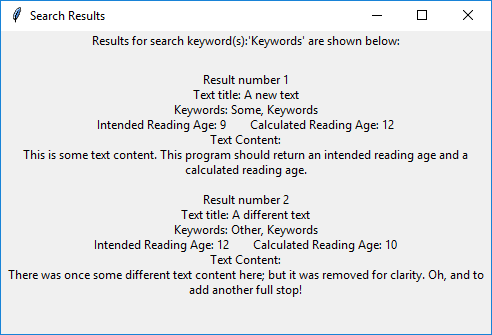
\* Testing evidence:



1



2



3

4

1. Using the calculator for calculation checking
2. Created ‘stored\_data.json’ file
3. ‘Typical’ inputs used
4. Search results

# Further Development

## Comparison against initial Solution Objectives

After the continuous testing process throughout and the testing documented above, my completed program does meet all of my initial objectives as I was able to improve and remove areas of erroneous code and further streamline some parts of the program. Referring back to my initial Solution Objectives, the final program…

* *does* *allow* the user to input text details and content
* *is able* to correctly count the number of characters, words and sentences in the given text content and thus calculate the text’s ARI
* *is able* to output the required data after processing and finally store this inputted and calculated data
* *does* *enable* the user to search through text data that was previously entered using either keywords or the calculated reading age
* *does* have a Graphical User Interface, for greatest ease of use, that is intuitive

## Potential Improvements

However, there are some aspects of the program that could be improved (although lots of them are not required by the objectives or initial ‘purpose’ and are merely areas that either improve overall coding quality or slightly enhance ease of use that I would have undertaken had there been more time). These areas include:

* Making the program DRYer (Don’t Repeat Yourself) by using python’s class inheritance system. This mainly applies to the GUI code as much of it is repeated at the beginning of every class (e.g. self.master, self.frame, self.title). The benefit of this is that the code would be somewhat easier to read and may run fractionally faster (though for a program of this size it should make little difference).
* In addition to the comments already in the final program, I could also have added docstrings for each function and method in order to make the code more self-documenting. This would improve readability (in addition to the existing comments) even further for a third-party.
* The organisation of the search results is, at times, confusing and hard to glean information from at a glance (especially when there are lots of results). Ideally, I would have implemented a scroll bar or displayed the results in a more tabulated format with distinctly separate columns for the category and data stored in each category for each individual text’s data.
* While not required by the brief or by my success criteria, a way to view the texts currently stored in the program (other than by just leaving the search box blank) could be a very useful feature. Additionally, it could even allow the user to edit certain details of a stored text that may have changed or been entered in incorrectly. This would require significant extension to the GUI – a plan for which is shown below.

