Report - Automated Readability Index

Scope of the Problem

Summary

The purpose of this project is to create a program that can calculate the reading age of a piece of text, using the ARI formula, and store the reading age, name of text, the text itself and any keywords, in a database that can be accessed by other teachers, so that teachers can store and locate texts based off of their keywords or reading ages.

The program should:

- enter and store the title of the text, the intended reading age, any keywords and the text's contents
- calculate the number of characters, words and sentences in the text
- use the ARI to calculate the reading age of the text
- store the calculated reading age of the text with the text details and text
- output the intended reading age and the calculated reading age of the text
- allow other teachers to search for the text using keywords and calculated reading age

Required Data

In order to do this, the program will require several inputs such as:

- the **title of the text**, which will allow the data to be stored and retrieved
- the **intended reading age of the text**, as an indicator of the reading age
- the **actual text** itself, which will be stored in a database, and be retrievable through the program
- any **keywords** relating to the text, to allow the text to be found easier by teachers
- **keywords** or **reading age** to search for, allowing the user to find texts from the database
- a **password** and **username**, entered by the user to ensure only staff are adding / accessing these texts
- the user's **choice**, to be used for menu navigation, and also to confirm decisions

The program will also need other data, that isn't input by the user, such as:

- the **ARI formula**, to allow the reading age to be calculated
    - *4.71 * (Characters / Words) + 0.5 * (Words / Sentences) – 21.43*
- the **number of characters in the text**, to be used in the ARI formula
- the **number of words in the text**, to be used in the ARI formula
- the **number of sentences in the text**, to be used in the ARI formula
- a correct **password** and **username**, for the user's ones to be checked against

These will either be pre-stored in the program, calculated from other inputs, or stored in files that can be accessed by the program.

Required Processes

The program will need to carry out some processes and calculations in order to work, such as:

- **calculate the number of characters in the text**, excluding spaces and punctuation, to be used in the ARI calculation
    - *set charcount = number of characters*
- **calculate the number of words in the text**, to be used in the ARI calculation
    - *set wordcount = number of spaces in text + 1*
- **calculate the number of sentences in the text**, to be used in the ARI calculation
    - *sentencecount = number of full stops + exclamation marks + question marks*
- **calculating the reading age of a text**, using the ARI formula (*and rounding the result up to the nearest integer*)
    - *set ARI = 4.71 * (Characters / Words) + 0.5 * (Words / Sentences) – 21.43*
- **calculate if the reading age is greater than, less than or equal to the intended reading age**, so that the program can inform the user if the text is easier/harder than intended
    - *set difference = intended reading age – actual reading age*
    - *if difference < 0, then difference = difference * -1*
- **store the text's contents in a file**, in order to be accessed later by other users
    - *title.write(text)*

- **store the text details** (calculated reading age, text title, keywords) with the text, in order to be accessed later
    - *title.write(keywords)*
- **search for other texts in the database**, using keywords or reading age, to allow them to find texts to use
    - *file.read()*
- **hash the passwords created by the program**, so that people cannot access them
- **de-hash the passwords created by the program**, so that the program can compare the user's input easier

The program will also need to do validation checks, to ensure it works smoothly, such as:

- **check that correct variable types have been input** (*title is a string, intended reading age is an integer, etc.*)
- **check that the password/ID that has been input is valid**, for security purposes
    - if password = "Password", then call program
    - else, call error
- **check that the user wants to perform certain tasks**, by making them input a choice and check if it is "*Yes*", or "*Y*", *etc.*

Required Outputs

The program will need to output some data in order to be seen and used by the user, these outputs will be:

- the **intended reading age of a text** along with its **actual calculated reading age**, and the difference between them, in order for the teacher to see if it is suitable for use and the intended difficulty
- the **text**, *along with its **title** and **reading age***, if it matches the user's search
- an **error message**, informing the user if something doesn't work (*such as it no results were found for their search*)
- **questions**, such as 'Would you like to search for a text, or create your own

entry?", to let the user know what they are doing

- a **GUI**, using things such as '===' or '***' to make the program look tidy and organised, and easier to understand and use
    - *==========*
    - *Title:*
    - *Reading Age:*
    - *==========*

Objectives

The objectives of the program are to:

- check if the user is a valid user by asking for an ID/password
- ask the user if they want to search for a text or add a new one, or add a new user to the system
- *if they choose to add a new text:*
    - allow the user to input data, such as the text's title, reading age, keywords and the text itself
    - calculate the number of characters, words and sentences in the text, to be used in the ARI equation
    - calculate the reading age using the ARI and the previously calculated data
    - store the calculated reading age of the text with the text, along with other text details such as the title and keywords
    - output the intended reading age and calculated reading age to show the teacher the accuracy of their estimation
- *if they choose to search for an existing text:*
    - allow for the user to search for a text based off of keywords and the reading age
    - display the desired text, *along with its title and reading age*, for the user to use in the class
- *if they choose to add a new user:*
    - allow the use to input a new username and password (verifying that the password is correct)
    - encrypt the data, and store it in a file that the program can access

later

Plan

Plan for the project:

Scope of the Problem	**1.5** hours
Design	**6** hours
Creating the Program	**7** hours
Testing	**3.5** hours
Evaluation	**2** hours

Design

Input and Output Facilities

Name	Data Type	Use	Validation
username	String	Used to check that the user can access the program. Authentication. Input.	Presence Check
password	String	Used to check that the user can access the program. Authentication. Input.	Presence Check
text	String	Used to calculate the ARI, and also stored in the database. Input and Output.	Presence Check
characters	Integer	Calculated from 'text' and is used in the ARI formula	Type Check, Presence Check
words	Integer	Calculated from 'text' and is used in the ARI formula	Type Check, Presence Check
sentences	Integer	Calculated from 'text' and is used in the ARI formula	Type Check, Presence Check
text_title	String	The title of the text, which the text will be stored under for easy use. Input and Output.	Presence Check
reading_age	Real	Will be calculated by the ARI formula and then rounded down, and then stored with the text, as well as shown to user. Output.	Type Check, Range Check, Presence Check
est_reading_age	Integer	Will be used to compare the user's estimated reading age with the actual reading age, and will be shown to user. Input and Output.	Type Check, Range Check, Presence Check
keyword	String	Keywords associated with the text to be stored with the text for easier searching. Input and Output.	Presence Check

choice	String	Will be used to determine if the user wants to search for an entry, or to add a new entry	Format Check, Presence Check
number_of_keywords	Integer	Used to cap how many times the program will loop the input of 'keyword'.	Format Check, Presence Check
new_username	String	Stored in a file to be used for future logins	Presence Check
new_password1	String	Stored in a file to be used for future logins	Presence Check
new_password2	String	Compared against new_password2 to check the user hasn't made any errors	Double Check, Presence Check

Data Structures

The program will need to store lots of data, and will do this through mainly two data structures: Arrays and Files. The program will store the keywords in an array. An array will be used to store all of the keywords in one place, this also reduces the amount of required variables (*rather than have 'keyword1', 'keyword2', etc., we can just have 'keyword', store it in the array, and then reuse it*).

0	**1**	**2**	**3**
keyword1	keyword2	keyword3	*etc.*

This will make storing the keywords more compact and easier to access. The program will then store the text, its title, the reading age, and its keywords in a file.

This will mean all of the data is stored in one place and will be easy to find and use, for example, when searching for texts, as only one file as to be searched, rather than multiple. This data will be stored into an array when the program is reading the file.

0	**1**	**2**	**3**
Line 1 (Title: title)	Line 2 (Text: text)	Line 3 (Reading Age: reading_age)	Line 4 (Keywords: keyword 1, etc.)

This will allow for the program to only print select parts of the file (e.g. only printing Line 2, the text itself), rather than printing the whole file, with all of the keywords. The program will also store all of the usernames and password in a file.

(The password is hashed, to prevent people from opening the file and reading it)

Validation Rules

The program will have to do validation to check if the inputs of the user are

acceptable for the program to function.

- **Presence Check** – check if the input of a variable is NULL or not, using a loop that will check if the variable has something in it.
    - *if variable = “”, then output “No data entered, please try again”, loop*
- **Range Check** – check if the data is within a reasonable range, and not to large, for example the ‘estimated reading age’ wouldn’t be any more than 18, and so user should not be able to input stupid values. The same applies with being under 5.
    - *if variable =>5, then if variable = <18, then continue*
- **Double Check** – check that the user has entered the correct data by making them input it twice, and comparing the two inputs. For example, with passwords.
    - *if password1 = password2, then continue*
- **Type Check** – check if the correct variable type has been used, for example, using an integer and not a string, as if a string is used in a calculation, it will not work properly.
    - *if variable ÷ variable = 1, then continue*
    - *if variable – variable = 0, then continue*
- **Format Check** – check that the variable is in the correct format.
- **Proof Reading** – check that no spelling, grammar or any other errors are present. The program itself cannot do this, but it can prompt the user to do it themselves
    - *Output “Please check that all information is correct, with no errors.”*

Inputs Designs

- **Forced Upper Case** – forcing certain variable entries to become upper case will make it easier for the use to operate the program, as they don’t have to worry about the case of their answer.

Outputs Designs

Calculated Reading Age:

Text:

Error: *(for example, a Presence Error)*

Functions

Authentication: The authentication used by this program will mainly be a password feature, that will make sure that the user is authorised to access the program, by making them enter a password, and checking it against a pre-existing password, which will be encrypted/hashed for added security.

Search: The search function will allow users to search for a text using keywords and or reading age, and will display valid texts.

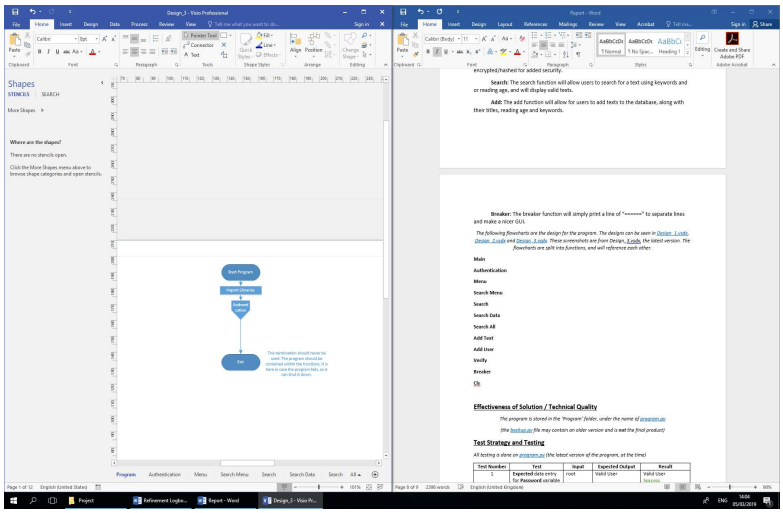
Add: The add function will allow for users to add texts to the database, along with their titles, reading age and keywords.

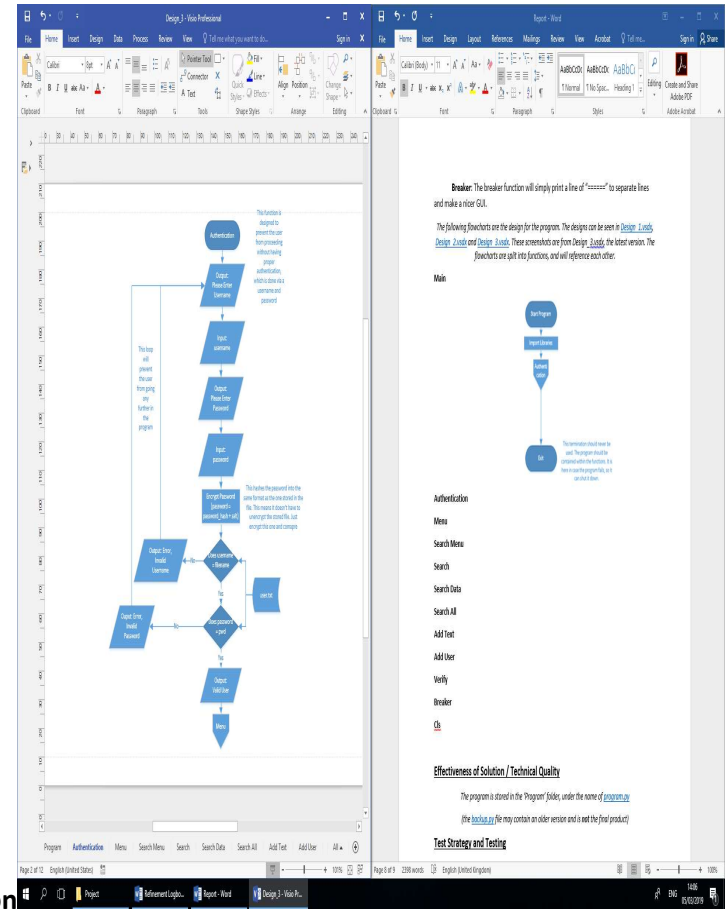
Breaker: The breaker function will simply print a line of "======" to separate lines and make a nicer GUI.

The following flowcharts are the design for the program. The designs can be seen in Design_1.vsdx, Design_2.vsdx and Design_3.vsdx. These screenshots are from Design_3.vsdx, the latest version. The flowcharts are split into functions, and will reference each other.

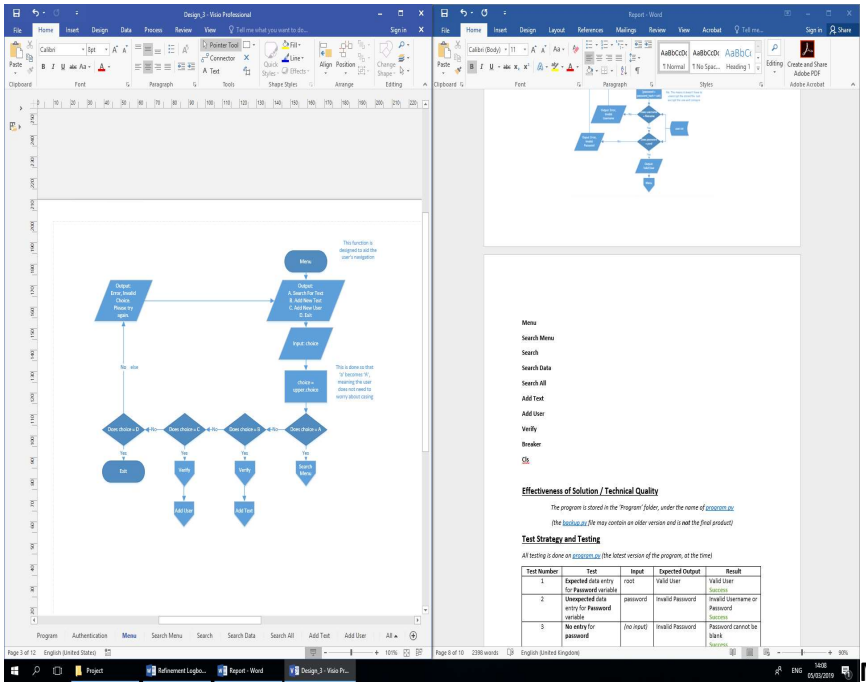
Please check Design_3.vsdx for more detail, and these screenshots may not be up to date if minor corrections (such as spelling) were made afterwards.

Main



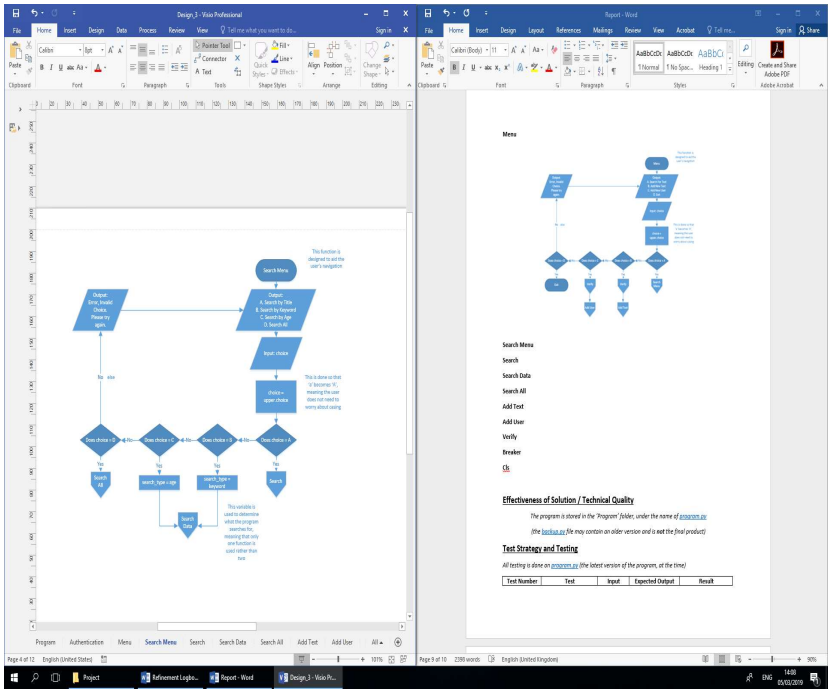


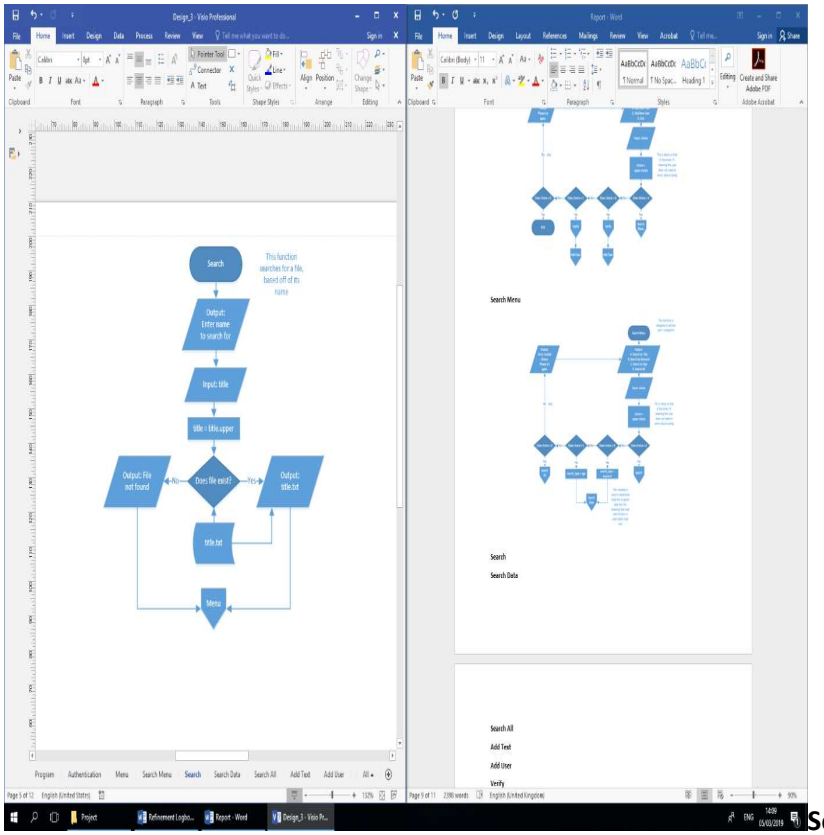
Authentication



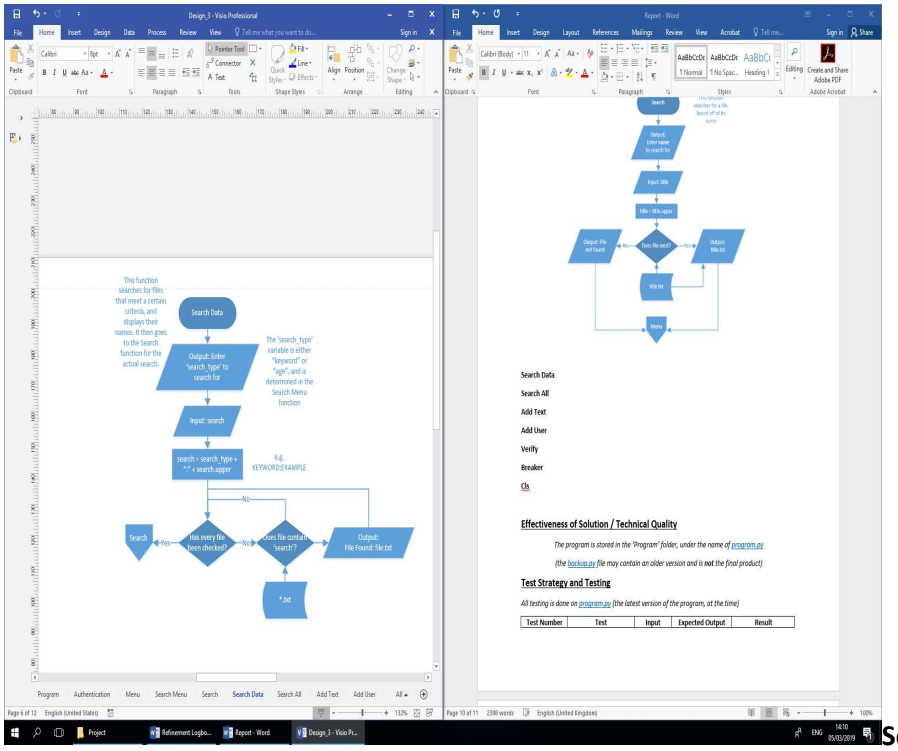
Menu

Search Menu

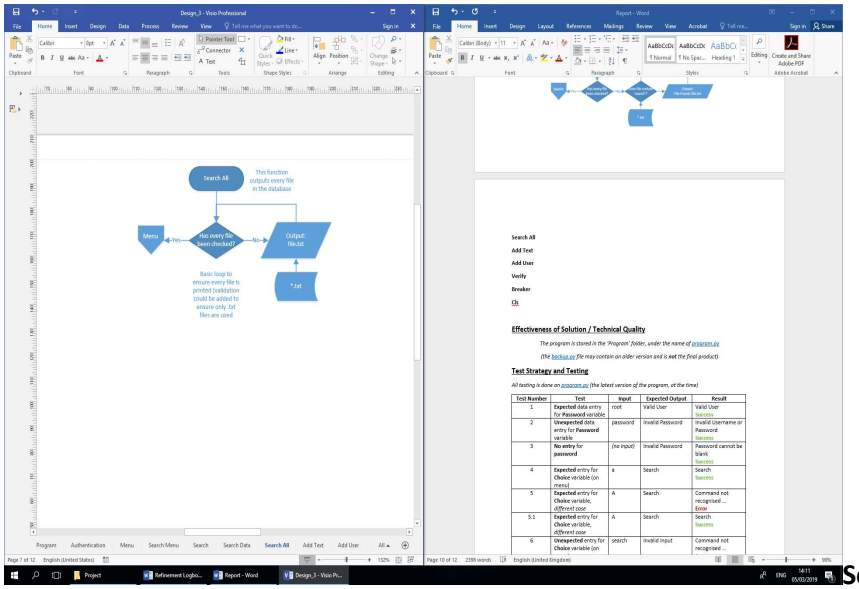

Menu
Search Menu
Search
Search Data
Search All
Add Text
Add User
Verify
Breaker
Cls
Effectiveness of Solution / Technical Quality
Test Strategy and Testing
Test Number
Test
Input
Expected Output
Result


Search
This function searches for a file, based off of its name
Output: Enter name to search for
Input: title
title = title.upper
Does file exist?
Output: File not found
Output: title.txt
title.txt
Menu
Search Menu
Search
Search Data
Search All
Add Text
Add User
Verify

Search

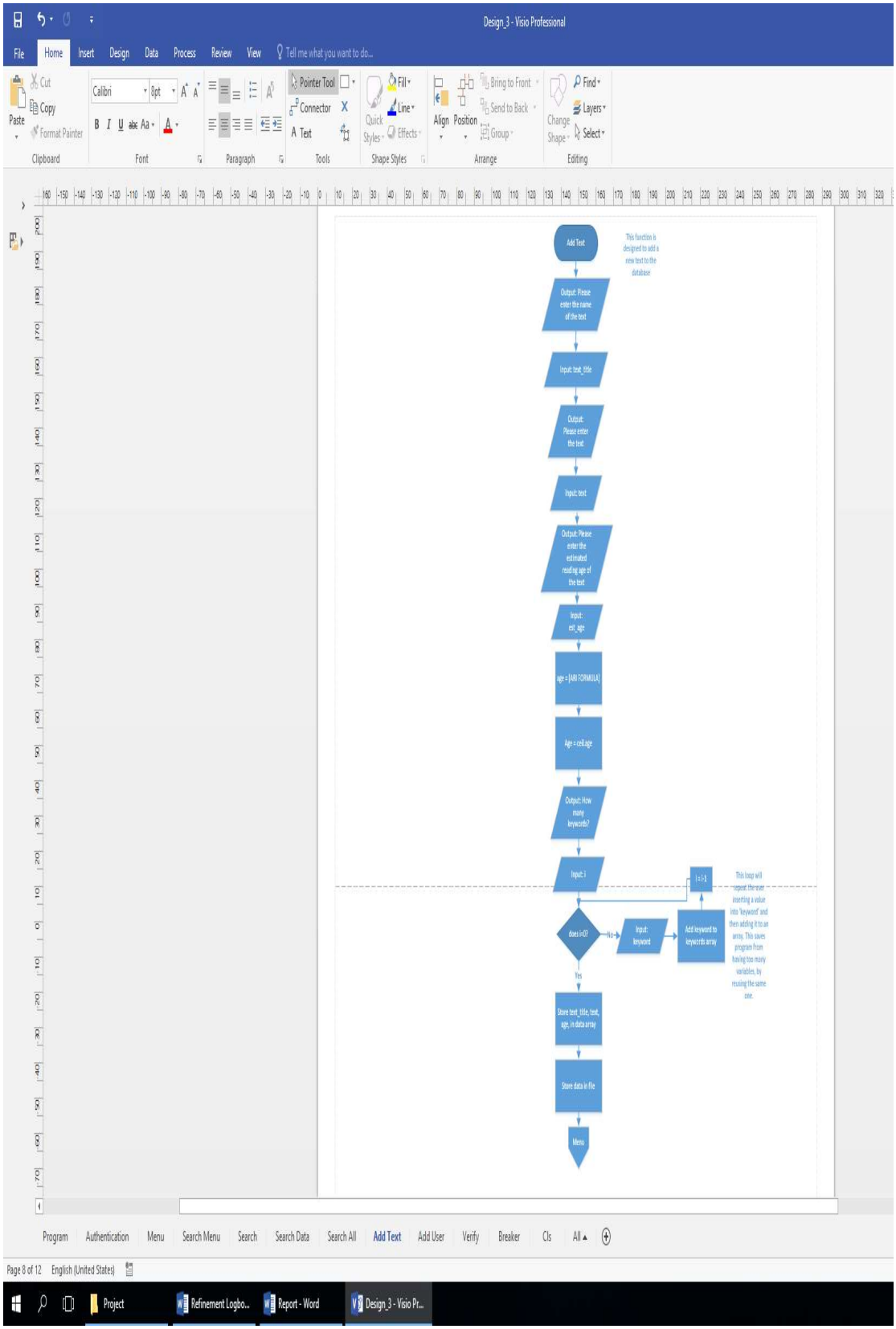


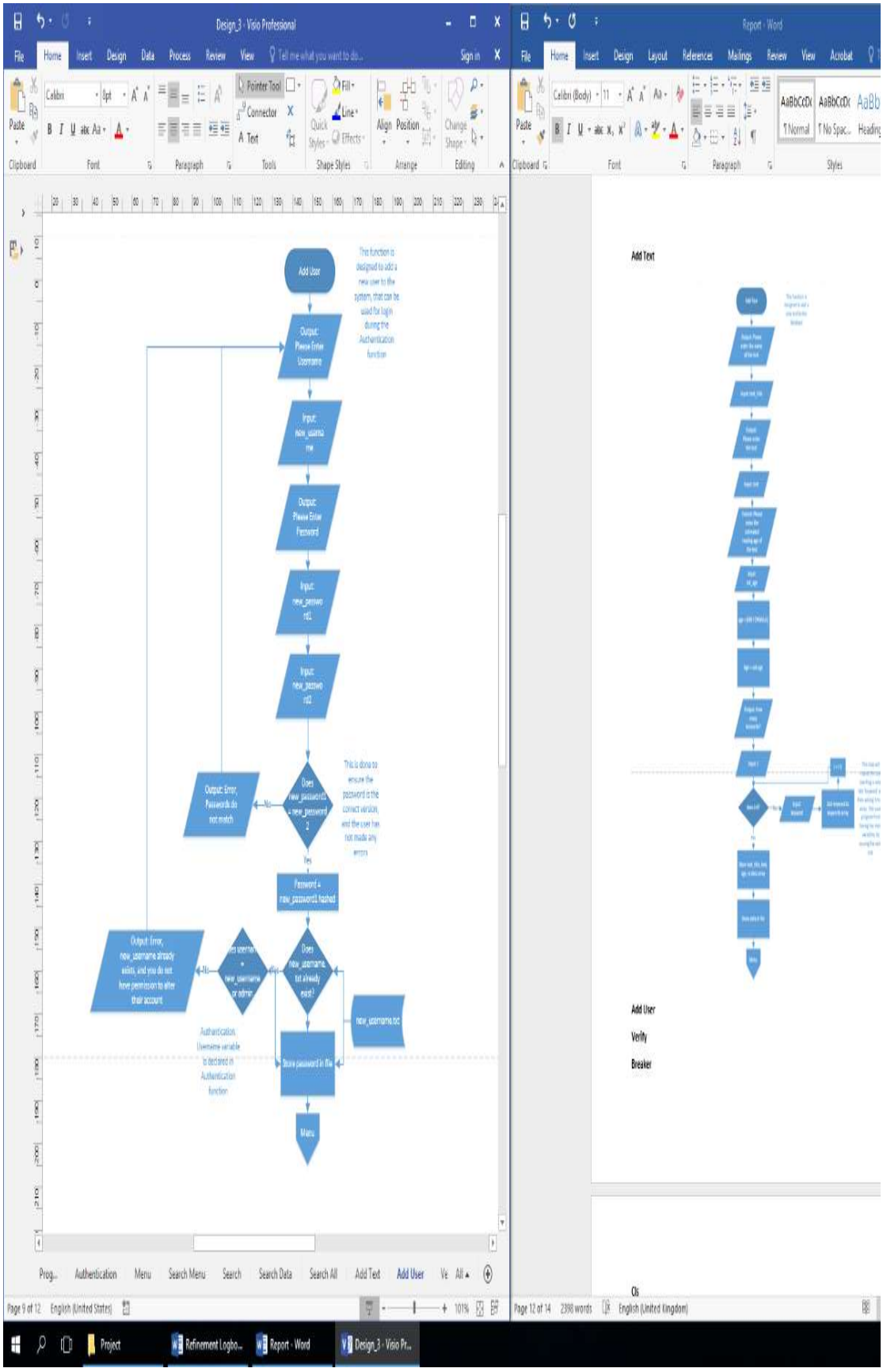
Search Data



Search All

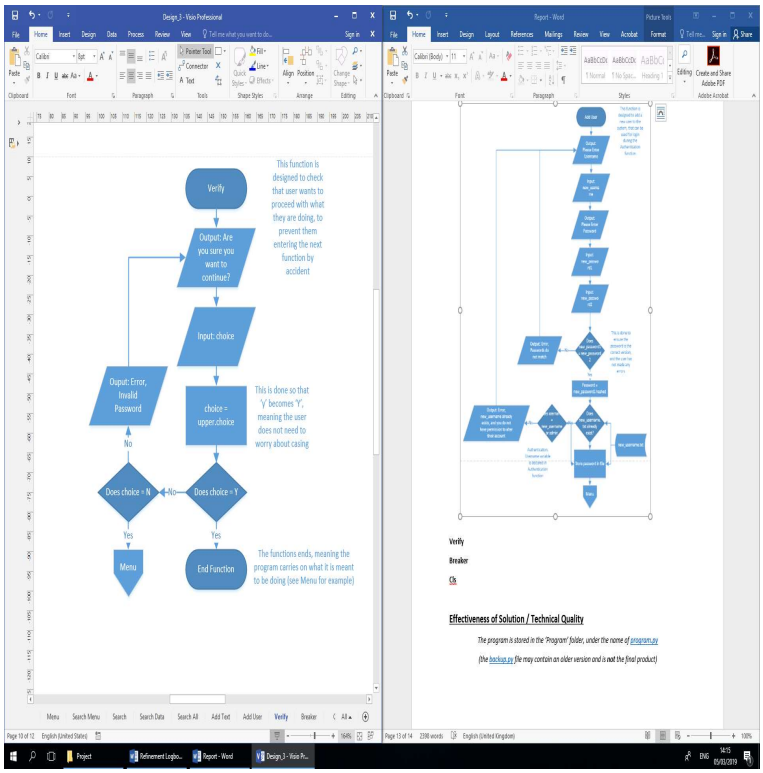
Add Text

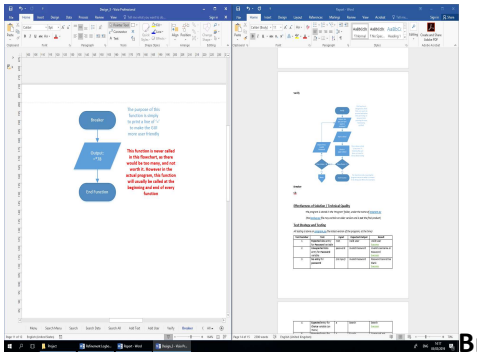

Design_3 - Visio Professional
File
Home
Insert
Design
Data
Process
Review
View
Tell me what you want to do...
Paste
Cut
Copy
Format Painter
Clipboard
Calibri
8pt
Font
Paragraph
Pointer Tool
Connector
Text
Tools
Quick Styles
Fill
Line
Effects
Shape Styles
Align
Position
Bring to Front
Send to Back
Group
Arrange
Change Shape
Find
Layers
Select
Editing
Add Text
This function is designed to add a new text to the database
Output: Please enter the name of the text
Input: text_title
Output: Please enter the text
Input: text
Output: Please enter the estimated reading age of the text
Input: est_age
age = [ARI FORMULA]
Age = ceil(Age
Output: How many keywords?
Input: i
i = i-1
does i=0?
No
Input: keyword
Add keyword to keywords array
This loop will repeat the user inserting a value into 'keyword' and then adding it to an array. This saves program from having too many variables, by reusing the same one.
Yes
Store text_title, text, age, in data array
Store data in file
Menu
Program
Authentication
Menu
Search Menu
Search
Search Data
Search All
Add Text
Add User
Verify
Breaker
Cls
All
Page 8 of 12
English (United States)
Project
Refinement Logbo...
Report - Word
Design_3 - Visio Pr...


Design_3 - Visio Professional
Report - Word
Add User
Output: Please Enter Username
Input: new_username
Output: Please Enter Password
Input: new_password1
Input: new_password2
Does new_password1 = new_password2
Output: Error, Passwords do not match
Password = new_password1.hashed
Output: Error, new_username already exists, and you do not have permission to alter their account
Does new_username.txt already exist?
new_username.txt
Store password in file
Menu
Add Text
Add User
Verify
Breaker

Add User

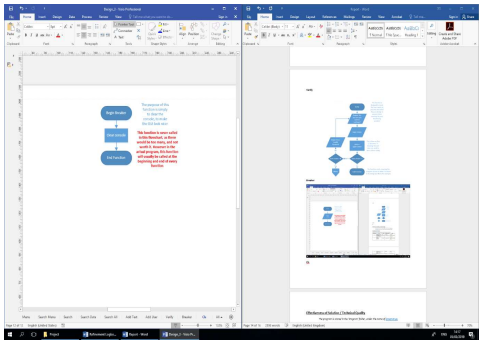
Verify



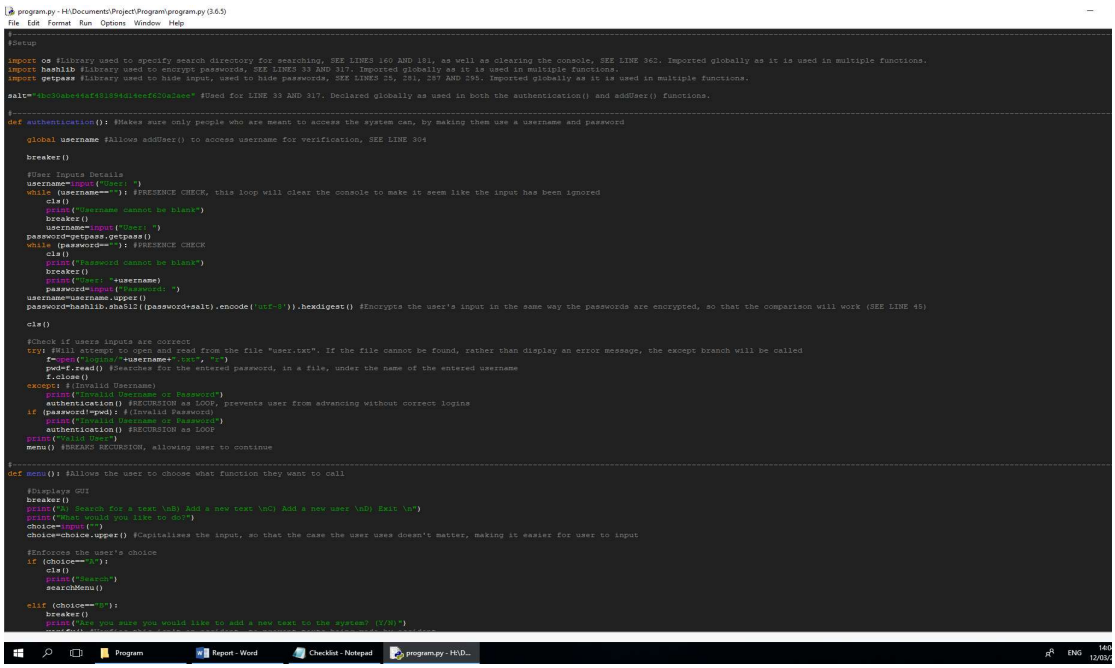


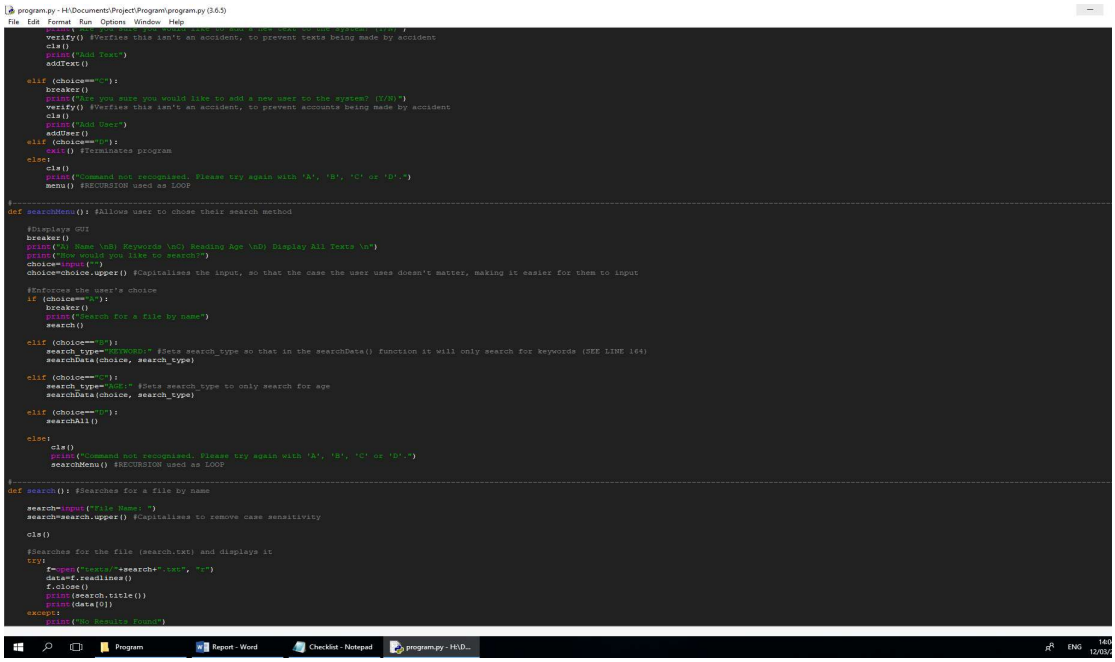
Breaker

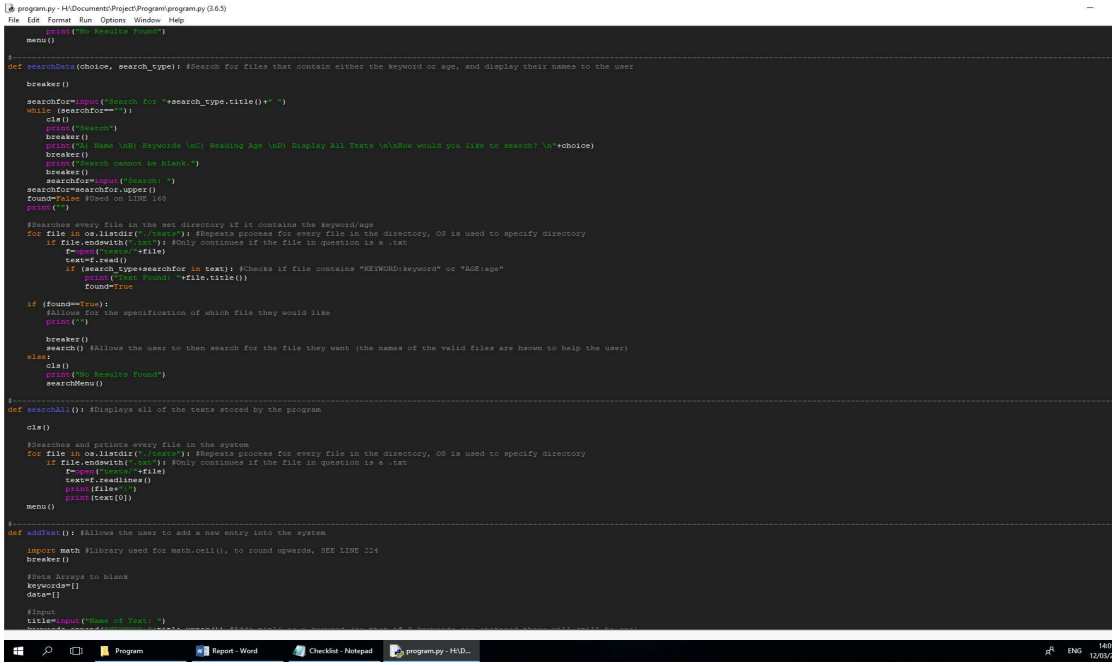
Cls



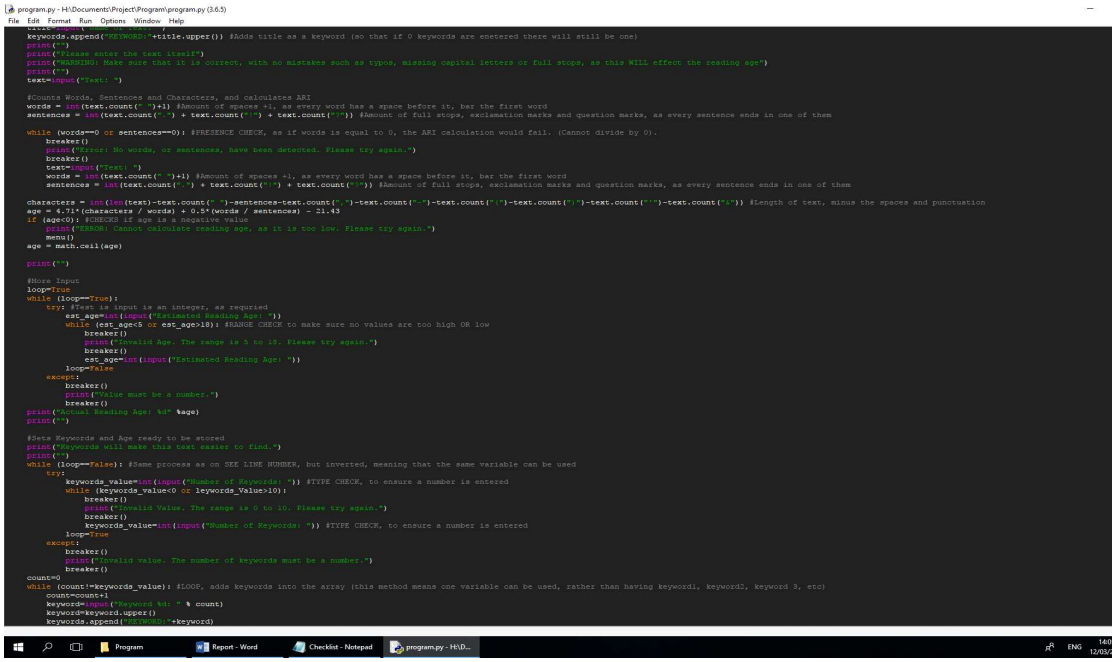
Effectiveness of Solution / Technical Quality


program.py - H:\Documents\Project\Program\program.py (3.6.5)
File Edit Format Run Options Window Help
#Setup
import os #Library used to specify search directory for searching, SEE LINES 160 AND 181, as well as clearing the console, SEE LINE 362. Imported globally as it is used in multiple functions.
import hashlib #Library used to encrypt passwords, SEE LINES 33 AND 317. Imported globally as it is used in multiple functions.
import getpass #Library used to hide input, used to hide passwords, SEE LINES 25, 251, 287 AND 295. Imported globally as it is used in multiple functions.
salt="4bc030abe46af481894d14eef620a2ace" #Used for LINE 33 AND 317. Declared globally as used in both the authentication() and addUser() functions.
def authentication(): #Makes sure only people who are meant to access the system can, by making them use a username and password
global username #Allows addUser() to access username for verification, SEE LINE 304
breaker()
#User Inputs Details
username=input("User: ")
while (username==""): #PRESENCE CHECK, this loop will clear the console to make it seem like the input has been ignored
cls()
print("Username cannot be blank")
breaker()
username=input("User: ")
password=getpass.getpass()
while (password==""): #PRESENCE CHECK
cls()
print("Password cannot be blank")
breaker()
print("User: "+username)
password=input("Password: ")
username=username.upper()
password=hashlib.sha512((password+salt).encode('utf-8')).hexdigest() #Encrypts the user's input in the same way the passwords are encrypted, so that the comparison will work (SEE LINE 45)
cls()
#Check if users inputs are correct
try: #Will attempt to open and read from the file "user.txt". If the file cannot be found, rather than display an error message, the except branch will be called
f=open("logins/"+username+".txt", "r")
pwd=f.read() #Searches for the entered password, in a file, under the name of the entered username
f.close()
except: #(Invalid Username)
print("Invalid Username or Password")
authentication() #RECURSION as LOOP, prevents user from advancing without correct logins
if (password!=pwd): #(Invalid Password)
print("Invalid Username or Password")
authentication() #RECURSION as LOOP
print("Valid User")
menu() #BREAKS RECURSION, allowing user to continue
def menu(): #Allows the user to choose what function they want to call
#Displays GUI
breaker()
print("A) Search for a text \nB) Add a new text \nC) Add a new user \nD) Exit \n")
print("What would you like to do?")
choice=input("")
choice=choice.upper() #Capitalises the input, so that the case the user uses doesn't matter, making it easier for user to input
#Enforces the user's choice
if (choice=="A"):
cls()
print("Search")
searchMenu()
elif (choice=="B"):
breaker()
print("Are you sure you would like to add a new text to the system? (Y/N)")
Program
Report - Word
Checklist - Notepad
program.py - H:\D...
ENG

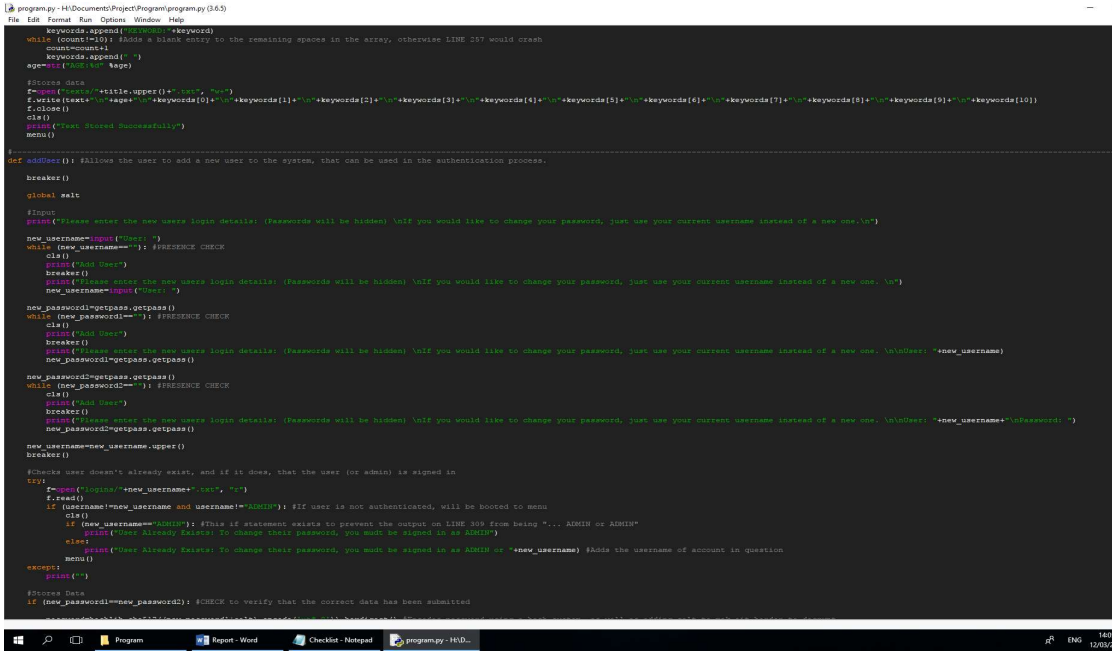

program.py - H:\Documents\Project\Program\program.py (3.6.5)
File Edit Format Run Options Window Help
verify() #Verfies this isn't an accident, to prevent texts being made by accident
cls()
print("Add Text")
addText()
elif (choice=="C"):
breaker()
print("Are you sure you would like to add a new user to the system? (Y/N)")
verify() #Verfies this isn't an accident, to prevent accounts being made by accident
cls()
print("Add User")
addUser()
elif (choice=="D"):
exit() #Terminates program
else:
cls()
print("Command not recognised. Please try again with 'A', 'B', 'C' or 'D'.")
menu() #RECURSION used as LOOP
def searchMenu(): #Allows user to chose their search method
#Displays GUI
breaker()
print("A) Name \nB) Keywords \nC) Reading Age \nD) Display All Texts \n")
print("How would you like to search?")
choice=input("")
choice=choice.upper() #Capitalisss the input, so that the case the user uses doesn't matter, making it easier for them to input
#Enforces the user's choice
if (choice=="A"):
breaker()
print("Search for a file by name")
search()
elif (choice=="B"):
search_type="KEYWORDS:" #Sets search_type so that in the searchData() function it will only search for keywords (SEE LINE 164)
searchData(choice, search_type)
elif (choice=="C"):
search_type="AGE:" #Sets search_type to only search for age
searchData(choice, search_type)
elif (choice=="D"):
searchAll()
else:
cls()
print("Command not recognised. Please try again with 'A', 'B', 'C' or 'D'.")
searchMenu() #RECURSION used as LOOP
def search(): #Searches for a file by name
search=input("File Name: ")
search=search.upper() #Capitalises to remove case sensitivity
cls()
#Searches for the file (search.txt) and displays it
try:
f=open("tests/"+search+".txt", "r")
data=f.readlines()
f.close()
print(search.title())
print(data[0])
except:
print("No Results Found")
Program
Report - Word
Checklist - Notepad
program.py - H:\D...
ENG

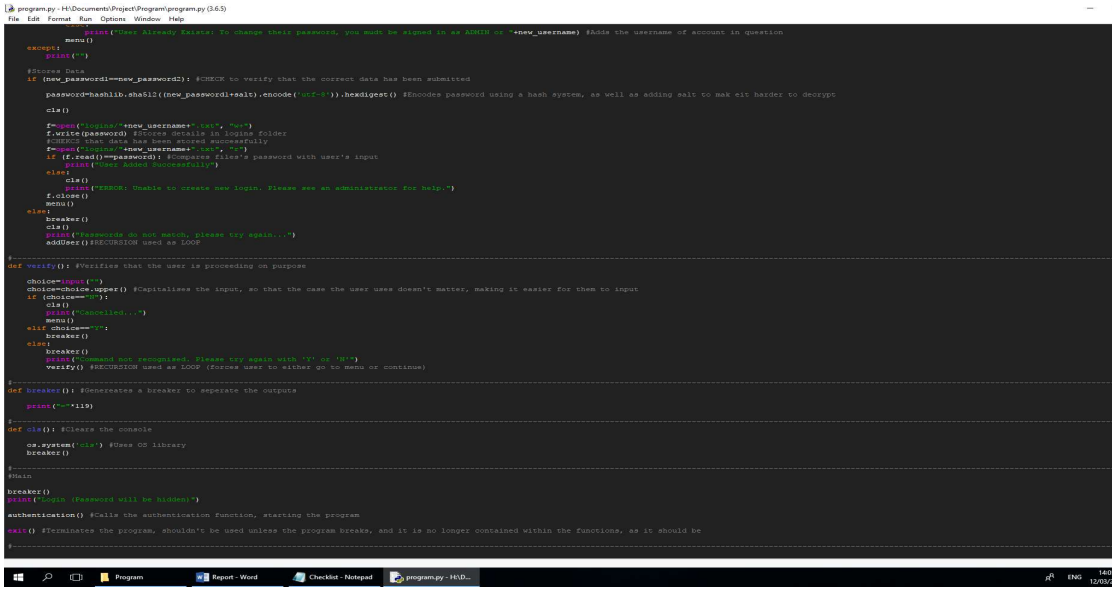


```
program.py - H:\Documents\Project\Program\program.py (3.6.5)
File  Edit  Format  Run  Options  Window  Help
        print("No Results Found")
    menu()

#-----------------------------------------------------------------------------------------------------------------------------------------------------------
def searchData(choice, search_type): #Search for files that contain either the keyword or age, and display their names to the user

    breaker()

    searchfor=input("Search for "+search_type.title()+" ")
    while (searchfor==""):
        cls()
        print("Search")
        breaker()
        print("A: Name \nB) Keywords \nC) Reading Age \nD) Display All Texts \n\nHow would you like to search? \n"+choice)
        breaker()
        print("Search cannot be blank.")
        breaker()
        searchfor=input("Search: ")
    searchfor=searchfor.upper()
    found=False #Used on LINE 168
    print("")

    #Searches every file in the set directory if it contains the keyword/age
    for file in os.listdir("./texts"): #Repeats process for every file in the directory, OS is used to specify directory
        if file.endswith(".txt"): #Only continues if the file in question is a .txt
            f=open("texts/"+file)
            text=f.read()
            if (search_type+searchfor in text): #Checks if file contains "KEYWORD:keyword" or "AGE:age"
                print("Text Found: "+file.title())
                found=True

    if (found==True):
        #Allows for the specification of which file they would like
        print("")

        breaker()
        search() #Allows the user to then search for the file they want (the names of the valid files are hsown to help the user)
    else:
        cls()
        print("No Results Found")
        searchMenu()

#-----------------------------------------------------------------------------------------------------------------------------------------------------------
def searchAll(): #Displays all of the texts stored by the program

    cls()

    #Searches and prints every file in the system
    for file in os.listdir("./texts"): #Repeats process for every file in the directory, OS is used to specify directory
        if file.endswith(".txt"): #Only continues if the file in question is a .txt
            f=open("texts/"+file)
            text=f.readlines()
            print(file+":")
            print(text[0])

    menu()

#-----------------------------------------------------------------------------------------------------------------------------------------------------------
def addText(): #Allows the user to add a new entry into the system

    import math #Library used for math.ceil(), to round upwards, SEE LINE 224
    breaker()

    #Sets Arrays to blank
    keywords=[]
    data=[]

    #Input
    title=input("Name of Text: ")
```

```
program.py - H:\Documents\Project\Program\program.py (3.6.5)
File  Edit  Format  Run  Options  Window  Help
    keywords.append("KEYWORD:"+title.upper()) #Adds title as a keyword (so that if 0 keywords are entered there will still be one)
    print("")
    print("Please enter the text itself")
    print("WARNING: Make sure that it is correct, with no mistakes such as typos, missing capital letters or full stops, as this WILL effect the reading age")
    print("")
    text=input("Text: ")

    #Counts Words, Sentences and Characters, and calculates ARI
    words = int(text.count(" "))+1) #Amount of spaces +1, as every word has a space before it, bar the first word
    sentences = int(text.count(".") + text.count("!") + text.count("?")) #Amount of full stops, exclamation marks and question marks, as every sentence ends in one of them

    while (words==0 or sentences==0): #PRESENCE CHECK, as if words is equal to 0, the ARI calculation would fail. (Cannot divide by 0).
        breaker()
        print("Error: No words, or sentences, have been detected. Please try again.")
        breaker()
        text=input("Text: ")
        words = int(text.count(" ")+1) #Amount of spaces +1, as every word has a space before it, bar the first word
        sentences = int(text.count(".") + text.count("!") + text.count("?")) #Amount of full stops, exclamation marks and question marks, as every sentence ends in one of them

    characters = int(len(text)-text.count(" ")-sentences-text.count(",")-text.count("-")-text.count("'")-text.count(")")-text.count("""")-text.count("&")) #Length of text, minus the spaces and punctuation
    age = 4.71*(characters / words) + 0.5*(words / sentences) - 21.43
    if (age<0): #CHECKS if age is a negative value
        print("ERROR: Cannot calculate reading age, as it is too low. Please try again.")
        menu()
    age = math.ceil(age)

    print("")

    #More Input
    loop=True
    while (loop==True):
        try: #Test is input is an integer, as required
            est_age=int(input("Estimated Reading Age: "))
            while (est_age<5 or est_age>18): #RANGE CHECK to make sure no values are too high OR low
                breaker()
                print("Invalid Age. The range is 5 to 18. Please try again:")
                breaker()
                est_age=int(input("Estimated Reading Age: "))
            loop=False
        except:
            breaker()
            print("Value must be a number.")
            breaker()
    print("Actual Reading Age: %d" %age)
    print("")

    #Sets Keywords and Age ready to be stored
    print("Keywords will make this text easier to find.")
    print("")
    while (loop==False): #Same process as on SEE LINE NUMBER, but inverted, meaning that the same variable can be used
        try:
            keywords_value=int(input("Number of Keywords: ")) #TYPE CHECK, to ensure a number is entered
            while (keywords_value<0 or keywords_value>10):
                breaker()
                print("Invalid Value. The range is 0 to 10. Please try again.")
                breaker()
                keywords_value=int(input("Number of Keywords: ")) #TYPE CHECK, to ensure a number is entered
            loop=True
        except:
            breaker()
            print("Invalid value. The number of keywords must be a number.")
            breaker()
    count=0
    while (count!=keywords_value): #LOOP, adds keywords into the array (this method means one variable can be used, rather than having keyword1, keyword2, keyword 3, etc)
        count=count+1
        keyword=input("Keyword %d: " % count)
        keyword=keyword.upper()
        keywords.append("KEYWORD:"+keyword)
```





The program is stored in the 'Program' folder, under the name of program.py (opening this link will run the program in the terminal, not open the IDE).

Please check program.py for more details, as the dark IDE makes the screenshots harder to read, also line numbers are referenced and cannot be viewed in the screenshots. Some minor chnages may have been made too (such as correcting spelling and grammar).

Test Strategy and Testing

All testing is done on program.py (the latest version of the program, at the time)

Test Number	Test	Input	Expected Output	Result
1	**Expected** data entry for **Password** variable	root	Valid User	Valid User **Success**
2	**Unexpected** data entry for **Password** variable	password	Invalid Password	Invalid Username or Password **Success**
3	**No entry** for **password**	*(no input)*	Invalid Password	Password cannot be blank **Success**
4	**Expected** entry for **Choice** variable (on menu)	A	Search	Search **Success**
5	**Expected** entry for **Choice** variable, *different case*	a	Search	Command not recognised … **Error**
5.1	**Expected** entry for **Choice** variable, *different case*	a	Search	Search **Success**
6	**Unexpected** entry for **Choice** variable (on menu)	search	Invalid Input	Command not recognised … **Success**
7	**Expected** entry for **Choice** variable (on search menu)	b	Search for Keyword	Search for Keyword **Success**
8	**Expected** entry for **Search** variable	test	Text Found: Test.txt	Text Found: Test.txt **Success**
9	**Unexpected** entry for **Search** variable	gandhi	No Results Found	*Program Ends* **Error**
9.1	**Unexpected** entry for **Search** variable	gandhi	No Results Found	No Results Found **Success**
10	**Expected** entry for **Search** variable (selecting text after searching via keyword)	test	Test The purpose of this project is …	Test The purpose of this project is … **Success**
11	**Extreme** entry for **Search** variable (selecting text after searching via keyword)	Test.txt	Test The purpose of this project is …	Test The purpose of this project is … **Success**
13	**Expected** entry for **Search** variable (selecting text after searching via keyword)	gandhi	Text Not Found	No Results Found **Success**
14	**Expected** entry for **Verify** function	N	[Menu]	Cancelled… [Menu] **Success**

15	**Unexpected** entry for **Text** variable (no full stop)	hello	No sentences detected	Error: No words, or sentences, have been detected. Please try again. *(loop)* **Success**
16	**Expected** entry for **Estimated Reading Age**	6	Actual Reading Age: …	Actual Reading Age: 3 **Success**
17	**Unexpected** entry for **Estimated Reading Age**	gandhi	Please enter a number	***Crashes*** **Error**
17.1	**Unexpected** entry for **Estimated Reading Age**	gandhi	Please enter a number	***Crashes*** **Error**
17.2	**Unexpected** entry for **Estimated Reading Age**	gandhi	Please enter a number	Value must be a number **Success**
18	**Unexpected** entry for **Number of Keywords**	string	Please enter a number *(loop)*	Please enter a number [menu] **Error**
18.1	**Unexpected** entry for **Number of Keywords**	string	Please enter a number *(loop)*	Invalid value. The number of keywords must be a number. **Success**
19	**Expected** entry for **Number of Keywords**	5	Keyword 1: … Keyword 2: … Keyword 3: … Keyword 4: … Keyword 5: … Text Stored Successfully	Keyword 1: … Keyword 2: … Keyword 3: … Keyword 4: … Keyword 5: … Text Stored Successfully **Success**
20	**Extreme** entry for **Number of Keywords**	11	Please choose a number between 0 and 10	Keyword 1: … Keyword 2: … … Keyword 11: … **Error**
20.1	**Extreme** entry for **Number of Keywords**	11	Please choose a number between 0 and 10	Invalid Value. The range is 0 to 10. Please try again. **Success**

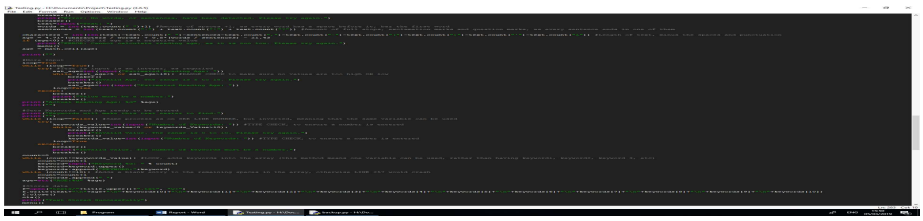
Changes made due to testing:

5

9

17

18 *and* **20**



Further Development

In conclusion, the program made meets all of the required objectives set at the beginning of the project. It can enter and store the title, reading age, keywords and the text content; calculate the number of characters, words and sentences in the text; use the ARI to calculate the actual reading age; store the text details, text and actual reading age in a file, output the intended reading age and calculated age of the text, and allow for other teachers to search for the text using keywords and reasing age.

On top of this, it can also do helpful functions that weren't mandatory. For example, an authentication system is in place to prevent unathorised access to the application. It has the ability to add new users as well as change the password of an existing user. Lots of validation is in place, for example, you can only change a password if you are signed in as said user, or as admin. This program can also output all texts and search by name, as well as just searching via keywords and age.

All functions in the program work as they should – as shown by the testing – and any improvement to them would be condensing the code (taking out unrequired lines, or replacing them with shorter way of doing the same task). This would make the code more efficient, and take up less storage, however, the code is already as efficient as I could make it, and it isn't excessivly large.

The program could be improved by adding extra features. For example, a way of recovering lost details could be helpful, otherwise forgetting a password means the user is shut out of the program with no way to get back in. Also, a way of deleting files within the program (as oppose to just deleting the text files in the folders) would be helpful and make the prgoram better. These shouldn't be too hard to program or to implement, there just wasn't enough time for me to do it though, and I didn't want to stray away from the primary task too much.

Notes for user

RUN IN CMD INTERFACE AND NOT THE IDLE! The cls() function won't work otherwise, and the breaker() function won't be fitted to the screen. Font should be SIZE 16 for breaker() function to work properly.

Default Login

User : admin

Password: root

Default Texts

Title : Test

Age : 16

Keywords: Test, Example

Title : Evaluation

Age : 16

Keywords: Evaluation, Report, Test

The ages (16) and keywords (Test) are the same to demonstrate how searching through for multiple items works.

Some comments reference lines numbers. The line numbers may not be exact, but are in close proximity (due to code changing). I've tried to correct them as much as possible but may have missed some.

The program will only allow texts of over 100 words. This is due to external research of the ARI formula, where I discovered this was required, as too short texts can generate negative reading ages.