**Title Page**

I confirm that this assignment is my own work. Where I have referred to academic sources, I have provided in-text citations and included the sources in the final reference list.

**Content Page:**

Section 1 Analysis…………………………………………………………………………………….3

Section 2 Design (Using an algorithm) …………………………………………………………….4

* Main Screen 4
* Login screen 5
* Student Screen 7
* Teacher Screen 8

Section 3 Technical Overview……………………………………………………………………...10

* General definitions 10
* ReadStudentAccounts.py section 13
* ReadTeacherAccounts.py section 13
* ViewUpdateExams.py section 14
* main.py 15

Section 4 Developing the coded solution…………………………………………………………20

* ReadStudentAccounts.py section 20
* ReadTeacherAccounts.py section 22
* ViewUpdateExams.py section 24
* main.py 25

Section 5 Testing to inform development…………………………………………………………40

* Database testing 40
* Student/Teacher login 41
* Teacher screen, update marks 42

Section 6 Testing to inform evaluation…………………………………………………………….43

* Student 43
* Teacher 46

Section 7 Evaluation of solution…………………………………………………………………...52

Section 8 References……………………………………………………………………………….55

Appendix ...………………………………………………………………………………….……….56

**Section 1: Analysis**

All data recorded is put into a database, such as admin, student and teacher accounts, subject students are taking and what exam results they have obtained. Performance wise it’s a lot better as high amounts of data can be stored in here, which means any number of students and teachers can be stored in the database, it is also easier to read and modify any data in the code. Plus, with a good database, it should be very organized and specific data can be located very easily through code.

The user interface system has a menu system to login in, students and teachers having a separate unique ID and password generated. The student UUID looks something like this: S000001, and the teacher UUID looks something like this: T000001. Once the user inputs their login credentials in accordance with whether they are a student or teacher, the login system will run a check in the code to identify them accordingly and bring them to their separate user programs. Whilst they are in the user programs, admins, students, and teachers can log out and it will send them back to the menu system.

In terms of permissions, students can only view their own exam results. For teachers, they are able to view specific student results and also view a subject’s list of students results, they are able to edit scores for the student’s assessment information. For admins, they have specific roles of adding new data into the database and deleting any unnecessary data in the database, only Robert will have this ability to do so, this allows scalability in the database by being able to update new students into the database with ease.

The exam names are stored under the database as CA1,SA1,CA2,SA2. It follows the method that Asian countries use to name their exams which are abbreviations called continual assessment and semesteral assessments which are held every quarter of the year accordingly.

There is a breakdown of the course information and the assessments in the course, plus students will be able to see a combined total for their course.

**Section 2: Design (Using an algorithm)**

**Main screen details:**

A diagram of a student login

Description automatically generated

def main\_screen\_choice():

If Option 1:

Student login

if Option 2:

Teacher login

**Student login**

A diagram of a student screen

Description automatically generated

def student\_login\_screen():

username = input(“Enter username here”)

password = input(“Enter password here”)

while True

if username == existing\_username:

if password == existing\_username\_password

Student Screen

else:

print(“Username entered incorrectly”)

else:

print(“Password entered incorrectly”)

**Teacher login**

**A diagram of a computer program

Description automatically generated**

def teacher\_login\_screen():

username = input(“Enter username here”)

password = input(“Enter password here”)

while True

if username == existing\_username:

if password == existing\_username\_password

Student Screen

else:

print(“Username entered incorrectly”)

else:

print(“Password entered incorrectly”)

**Student Screen**

A diagram of a flowchart

Description automatically generated

def display\_student\_data():

student = username

Check database

Retrieve all “student” exam marks in all 4 subjects

Display data on student screen

def log\_out:

Send user back to main screen

**Teacher Screen**

A diagram of a student's program

Description automatically generated

def subject\_or\_student():

subject = [“English”, “Math”, “Chinese”, “Science”]

get\_input = input(“Subject or student”)

if get\_input in subject:

Check database

Retrieve all student exam marks of subject

Display data on teacher screen

If get\_input == existing\_username

Check database

Retrieve all student’s subject exams and marks

Display data on teacher screen

def update\_marks():

subjects = [“English”, “Math”, “Chinese”, “Science”]

student\_id = input(“Student ID”)

subject = input(“Subject”)

ca1 = input(“Update CA1 marks”)

sa1 = input(“Update SA1 marks”)

ca2 = input(“Update CA2 marks”)

sa2 = input(“Update SA2 marks”)

while True:

if student\_id == existing\_username:

if subject in subjects:

if ca1,sa1,ca2,sa2 in range (0,100):

Check database

Update all student’s exam marks of subject

Send data back to database

break

else:

continue

else:

continue

else:

continue

def log\_out:

Send user back to main screen

**Section 3: Technical Overview**

**Definitions**

**Python**

list: a mutable, ordered data structure that can store multiple items in a single instance, mutable data structures allow the user to modify the data structure’s internal states even after its been created (Python lists, no date).

tuple: immutable, ordered data structure that can store multiple items in a single variable, they are more memory-efficient than lists (Python tuples, no date).

.append(): append stores the variable and applies it to the end of the list (Python List append() Method, no date).

for loop: runs the block of code until conditions are no longer met and it promptly breaks and continues to the next line of code (Python For loops, no date).

If statement is a control structure that will execute different blocks of code depending on the conditions set.

return: ends the function call and returns the result back to the user (Python return Keyword, no date).

Global: allows access to the variable outside of the function, as everything in a function is local and other functions cannot read the variable without global (Python – Global Variables, no date).

**Class**

class: It is an object constructor that provides a skeletal structure or blueprint that allows data to be inserted and create an object (What is a constructor in Python? , no date).

def(): function that runs the block of code when called, allows data to be passed into the function under (), and can return data when necessary. It is reusable in many instances and different sets of data or parameters can be put under the () to produce different outputs (Python Functions, no date).

def \_\_init\_\_(self,uuid,name,password):

* \_\_init\_\_: creates object properties for the data to be stored in.
* self: represents the instance of a class name, initializes object attributes.
* uuid,name,password: object attributes for variable data to be stored in

**Sqlite**

sqlite3.connect(): allows python to connect to the local database browser sqlite3. It can only pull databases from under the same folder directory it calls from in the code ([sqlite3](https://docs.python.org/3/library/sqlite3.html#module-sqlite3) — DB-API 2.0 interface for SQLite databases, no date).

.cursor: gives the user a temporary workstation and is created by these database browsers to allow data manipulation operations, it reads data from the database row-by-row and can iterate over a result set through operations such as using for loops or any type of recursive queries. It allows other complex operations too such as pulling data from multiple tables as this is important for processing databases with hierarchical data, structures and can perform multiple operations that will be useful such as updating data, selecting data, and inserting new data ([sqlite3](https://docs.python.org/3/library/sqlite3.html#module-sqlite3) — DB-API 2.0 interface for SQLite databases, no date).

cursor.execute(): executes a single line of SQL statement based on variable set inside ([sqlite3](https://docs.python.org/3/library/sqlite3.html#module-sqlite3) — DB-API 2.0 interface for SQLite databases, no date).

cursor.fetchall(): pulls all rows of data ([sqlite3](https://docs.python.org/3/library/sqlite3.html#module-sqlite3) — DB-API 2.0 interface for SQLite databases, no date).

cursor.close: closes the connection to the database ([sqlite3](https://docs.python.org/3/library/sqlite3.html#module-sqlite3) — DB-API 2.0 interface for SQLite databases, no date).

In sqlite ? is a placeholder for a real value

**TKinter**

Tk(): creates an instance of tk frame, it is usually seen as the parent screen (tkinter – Python interface to Tk, no date).

.geometry(): creates the size of the parent screen (tkinter – Python interface to Tk, no date).

.title(): Gives the parent screen a title at the top left of the application (tkinter – Python interface to Tk, no date).

Widget: python object, instantiated into class e.g ttk.Label, ttk.Entry, ttk.Button (tkinter – Python interface to Tk, no date).

Label(): is a widget that allows user to implement text or images in a display box. The main syntaxes used throughout the code are screen, text, background, width, height, font. These customises the display box accordingly to what the syntaxes are (tkinter – Python interface to Tk, no date).

.pack(): is a feature that organises these widgets into blocks that pastes itself onto the parent window, there is also syntax within .pack that allows you to pack it in the cardinal directions and will pack itself in the specified cardinal direction, it also packs any widget from top-down as that is how python processes code (tkinter – Python interface to Tk, no date).

Radiobutton(): is a widget that allows user to pick between a multiple-choice option, likewise with label it also has syntax used to screen and text, plus a variable and value that creates this multiple-chocie option (tkinter – Python interface to Tk, no date).

Button(): Is a widget that creates a button for the user to press, has the syntax of screen, text, width, height and command. Likewise with the other widgets with the exception of the command, it has similar functions (tkinter – Python interface to Tk, no date).

.mainloop: runs tkinter event in a loop

Toplevel(): is a widget that just applies the window on top of all the other windows, in this case it will apply itself on top of the main\_screen (tkinter – Python interface to Tk, no date).

Entry(): is a widget that accepts one line text strings from the user. By storing it in a textvariable, it is possible to return the text written by the user by storing it as a StringVar() and using .get() to convert it into a string (tkinter – Python interface to Tk, no date).

StringVar(): holds a string (Variable classes in tkinter, 2021)

.destroy() removes the window from the page and all events under the window is promptly removed (tkinter – Python interface to Tk, no date).

**Treeview**

Ttk.TreeView() is a widget that utilizes a hierarchical structure to display multiple items and has in built functions such as scrolling up and down to allow easy access for users to browse through it. Some of the syntaxes used are parent screen, columns, and headings (Treeview, no date).

tree.column() is used to identify columns based on the tree id written, such as c1, c2, c3, c4, c5. The syntax used are id, minwidth, width, anchor and stretch.

tree.heading() is used to display text under the columns identified as its syntax are column id and text.

**Variables used:**

**ReadStudentAccounts.py**

def get\_all\_students\_from\_db(): Function that obtains all the student’s uuid name and password from the database students.db3 under the table StudentAccounts

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| students | List | store all the student’s uuid, name and password from the StudentAccounts table |
| sqliteConnection | Variable | connects to database ‘students.db3’ |
| cursor | Variable | connects to the database 'students.db3' and allows user to data manipulate anything inside the database through the temporary workstation |
| sqlite\_select\_query | String | Pulls table name StudentsAccounts and columns of data under the UUID, Name, Password sections. |
| records | List | pulls all rows of data under the StudentAccounts table |
| row | Tuple | stores the data pulled from records. |
| student | Student object | Stores the class data of Student. |
| row[0], row[1], row[2] | Tuple elements | Store the column data UUID, Name, Password from StudentAccounts accordingly and applies the data into the object attributes of Student. |

**ReadTeacherAccounts.py**

def get\_all\_teachers\_from\_db(): Function that obtain all the teacher's uuid name and password from the database students.db3 under the table TeacherAccounts

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| teachers | List | store all the teacher’s uuid, name and password from the TeacherAccounts table |
| sqliteConnection | Variable | connects to database ‘students.db3’ |
| cursor | Variable | connects to the database 'students.db3' and allows user to data manipulate anything inside the database through the temporary workstation |
| sqlite\_select\_query | String | Pulls table name TeacherAccounts and columns of data under the UUID, Name, Password sections. |
| Records | List | pulls all rows of data under the TeacherAccounts table |
| row | Tuple | stores the data pulled from records. |
| teacher | Teacher object | Stores the class data of Teacher. |
| row[0], row[1], row[2] | Tuple elements | Store the column data UUID, Name, Password from TeacherAccounts accordingly and applies the data into the object attributes of Teacher. |

**ViewUpdateExams.py**

def update\_multiple\_columns(StudentUUID, Subject, CA1, SA1, CA2, SA2): Function is used to update student's exam marks in the database.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| sqliteConnection | Variable | connects to database ‘students.db3’ |
| cursor | Variable | connects to the database 'students.db3' and allows user to data manipulate anything inside the database through the temporary workstation |
| sqlite\_update\_query | String | Stores the Subject positional argument variable into the string and updates the subject’s exam marks in accordance with the 4 exam positional arguments plus the Student UUID positional argument when executed |

**Main.py**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| list\_of\_subjects, list\_of\_exams,  students,  teachers | List | Preloaded lists in the code to allow easy access for functions later in the code. |

def log\_student\_or\_teacher(): Picks function, login verification, based on radio button clicked.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| val | Int | Obtains the IntVar() of var and stores the integer number |

def subject\_or\_student\_marks(): Picks function based on what is inputted, subject or student UUID.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| subject\_or\_student | String | Gets the StringVar() subject\_or\_mark and turns it to a string |
| s | Tuple | iterates over the list, students that contain the Student class object attributes in the students list through a for loop |
| students | List | store all the student’s uuid, name and password from the StudentAccounts table |

def get\_student\_marks\_stscreen(): Get all marks of one student in stscreen

def get\_student\_marks(): Get all marks of one student teacherscreen

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| sqliteConnection | Variable | connects to database ‘students.db3’ |
| cursor | Variable | connects to the database 'students.db3' and allows user to data manipulate anything inside the database through the temporary workstation |
| results | List | store all the student’s exam results from CA1, SA1, CA2, SA2 from all 4 subjects English, Math, Chinese, Science |
| x | Tuple | iterates over the list, list\_of\_subjects. |
| query | String | Selects the table in order of the list\_of\_subjects, which then selects the exam results of the StudentUUID acquired |
| records | List | pulls all rows of data under the StudentAccounts table |
| result | Tuple | iterates over the list, results. |

def subject\_exam\_results(): # Gets all student marks in one subject.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| sqliteConnection | Variable | connects to database ‘students.db3’ |
| cursor | Variable | connects to the database 'students.db3' and allows user to data manipulate anything inside the database through the temporary workstation |
| sqlite\_select\_query | String | Selects all of the data from the table name based on what is in the string variable subject\_or\_student |
| records | List | Fetches all student exam results under the table of the string variable subject\_or\_student |

def student\_login\_verify(): Used to check username\_verify and password\_verify if they are a student and identify if the login credentials exist in the database.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| username1 | String | Obtains the StringVar() of username\_verify and will store the string. |
| password1 | String | Obtains the StringVar() of password\_verify and will store the string into the variable password1 |
| student | Variable | Will store inputted tuple data of s |
| s | Tuple | iterates over the list, students that contain the Student class object attributes in the students list through a for loop |
| student\_name | Variable | Store the student.name as the variable student\_name |
| student\_uuid | Variable | Store the student.uuid as the variable student\_uuid |

def teacher\_login\_verify(): Used to check username\_verify and password\_verify if they are a teacher and identify if the login credentials exist in the database.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| username1 | String | Obtains the StringVar() of username\_verify and will store the string. |
| password1 | String | Obtains the StringVar() of password\_verify and will store the string into the variable password1 |
| teacher | Variable | Will store inputted tuple data of t |
| t | Tuple | iterates over the list, students that contain the Teacher class object attributes in the teachers list through a for loop |
| teacher\_name | Variable | Store the teacher.name as the variable teacher\_name |
| teacher\_uuid | Variable | Store the teacher.uuid as the variable teacher\_uuid |

def update\_marks(): Update all marks of one student in one subject into the database.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| UUID | String | Obtains the StringVar() of UUID\_Update and will store the string. |
| subject | String | Obtains the StringVar() of Subject\_Update and will store the string. |
| s | Tuple | iterates over the list, students that contain the Student class object attributes in the students list through a for loop |
| student | Variable | Will store inputted tuple data of s |
| ca1 | Integer | Obtains the IntVar() of CA1\_marks and store the integer. |
| sa1 | Integer | Obtains the IntVar() of SA1\_marks and store the integer. |
| ca2 | Integer | Obtains the IntVar() of CA2\_marks and store the integer. |
| sa2 | Integer | Obtains the IntVar() of SA2\_marks and store the integer. |

def password\_not\_recognised(): Designing popup for login invalid password.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| password\_not\_recog\_screen | Toplevel | Places password\_not\_recog\_screen on top of login screen |

def user\_not\_found(): Designing popup for user not found.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| user\_not\_found\_screen | Toplevel | Places user\_not\_found\_screen on top of login screen |

def subject\_not\_found(): Designing popup for subject not found.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| subject\_not\_found\_screen | Toplevel | Places subject\_not\_found\_screen on top of login screen |

def marks\_not\_found(): Designing popup for subject not found.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| marks\_not\_found\_screen | Toplevel | Places marks\_not\_found\_screen on top of login screen |

def update\_successful(): Designing popup for marks being successfully updated.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| update\_successful\_screen | Toplevel | Places update\_successful\_screen on top of updatemarksscreen |

def log\_out(): Designing Log out Button.

def clear\_Tree(): Designing clear all TreeView Button for tree

def clear\_Tree2(): Designing clear all TreeView Button for tree2

def delete\_password\_not\_recognised(): Deleting popups.

def delete\_user\_not\_found\_screen(): Deleting popups.

def delete\_subject\_not\_found\_screen(): Deleting popups.

def delete\_marks\_not\_found\_screen(): Deleting popups.

def delete\_update\_successful\_screen(): Deleting popups

def main\_account\_screen(): Designing Main(first) window.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| main\_screen | Tk | Parent screen |
| var | Integer data | Gets radiobutton integer data |

def login(): Designing window for login.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| main\_screen | Tk | Parent screen |
| login\_screen | Toplevel | Places the login\_screen on top of the main\_screen |
| username\_verify | String data | Gets entry from username\_verify and converts it to string data |
| password\_verify | String data | Gets entry from password\_verify and converts it to string data |

def student\_screen(): Designing student screen.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| stscreen | Toplevel | Places the stscreen on top of the login\_screen |
| Tree | Treeview | Displays all the data of the student’s marks |

def teacher\_screen(): Designing teacher screen.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| teacherscreen | Toplevel | Places the teacherscreen on top of the login\_screen |
| subject\_or\_mark | String data | Gets entry from subject\_or\_mark and converts it to string data |
| Tree2 | Treeview | Displays variant data for the teacher to see |
| i | Integer | Display the number of iterations through the loop. |
| subject | String | Display what subject is being iterated over currently. |

def update\_marks\_screen(): Obtain entries to update marks in database

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Purpose** |
| Updatemarksscreen | Toplevel | Places the updatemarksscreen on top of the teacherscreen |
| UUID\_Update | String data | Gets entry from UUID\_Update and converts it to string data |
| Subject\_Update | String data | Gets entry from Subject\_Update and converts it to string data |
| CA1\_marks | Integer data | Gets entry from CA1\_marks and converts it to integer data |
| SA1\_marks | Integer data | Gets entry from SA1\_marks and converts it to integer data |
| CA2\_marks | Integer data | Gets entry from CA2\_marks and converts it to integer data |
| SA2\_marks | Integer data | Gets entry from SA2\_marks and converts it to integer data |

**Section 4: Developing the coded solution:**

**Under ReadStudentAccounts.py**

import sqlite3

class Student:

    def \_\_init\_\_(self,uuid,name,password):

        # Object Attributes

        self.uuid = uuid

        self.name = name

        self.password = password

def get\_all\_students\_from\_db(): # Function has been adapted from https://pynative.com/python-sqlite-select-from-table/

    # Function that obtain all the student’s uuid name and password

    # from the database students.db3 under the table StudentAccounts

    students = []

    # students is a list

    sqliteConnection = sqlite3.connect('students.db3')

    # Connects to database ‘students.db3’ and stores it in variable sqliteConnection.

    cursor = sqliteConnection.cursor()

    # Connects to the database 'students.db3' and allows user to data manipulate anything

    # inside the database through the temporary workstation, stores it in a variable called cursor.

    sqlite\_select\_query = """SELECT UUID,Name,Password from StudentAccounts"""

    # Literal string stored in variable sqlite\_select\_query, pulls table name StudentsAccounts

    # and columns of data under the UUID, Name, Password sections.

    cursor.execute(sqlite\_select\_query)

    # Executes variable string name sqlite\_select\_query through the cursor.

    records = cursor.fetchall()

    # Pulls all rows of data under the StudentAccounts table and stores it as a list under the list records.

    for row in records: # For loop repeats until no more rows of data can be pulled from StudentAccounts table

    # row is a tuple that stores the data pulled from records.

        student = Student(row[0],row[1],row[2])

        # student is a variable that stores the class data of Student.

        # row[0], row[1], row[2] are the 3 elements in the tuple that store the column data UUID, Name, Password

        # from StudentAccounts accordingly and applies the data into the object attributes of Student.

        students.append(student)

        # Appends the variable student, applies it to the end of the list students

    cursor.close()

    # Closes the connection to the database

    return students

    # Returns the list students and the final output of the function get\_all\_students\_from\_db() is all the student’s

    # uuid, name and password from the StudentAccounts table and is displayed as the list, students.

def unit\_test():

    students = get\_all\_students\_from\_db()

    assert(len(students) > 0)

unit\_test()

**Under ReadTeacherAccounts.py**

import sqlite3

class Teacher:

    def \_\_init\_\_(self,uuid,name,password):

        # Object Attributes

        self.uuid = uuid

        self.name = name

        self.password = password

def get\_all\_teachers\_from\_db(): # Function has been adapted from https://pynative.com/python-sqlite-select-from-table/

    # Function that obtain all the teacher's uuid name and password

    # from the database students.db3 under the table TeacherAccounts

    teachers = []

    # teachers is a list

    sqliteConnection = sqlite3.connect('students.db3')

    # Connects to database ‘students.db3’ and stores it in variable sqliteConnection.

    cursor = sqliteConnection.cursor()

    # Connects to the database 'students.db3' and allows user to data manipulate anything

    # inside the database through the temporary workstation, stores it in a variable called cursor.

    sqlite\_select\_query = """SELECT UUID,Name,Password from TeacherAccounts"""

    # Literal string stored in variable sqlite\_select\_query, pulls table name TeacherAccounts

    # and columns of data under the UUID, Name, Password sections.

    cursor.execute(sqlite\_select\_query)

    # Executes variable string name sqlite\_select\_query through the cursor.

    records = cursor.fetchall()

    # Pulls all rows of data under the TeacherAccounts table and stores

    # it as a list under the list records.

    for row in records: # For loop repeats until no more rows of data can be pulled from TeacherAccounts table

    # row is a tuple that stores the data pulled from records.

        teacher = Teacher(row[0],row[1],row[2])

        # teacher is a variable that stores the class data of Teacher.

        # row[0], row[1], row[2] are the 3 elements in the tuple that store the column data UUID, Name, Password

        # from TeacherAccounts accordingly and applies the data into the object attributes of Teacher.

        teachers.append(teacher)

        # Appends the variable teacher, applies it to the end of the list teachers

    cursor.close

    # Closes the connection to the database

    return teachers

    # Returns the list teachers and the final output of the function get\_all\_teachers\_from\_db() is all the teacher’s

    # uuid, name and password from the TeacherAccount table and is displayed as the list, teachers.

def unit\_test():

    teachers = get\_all\_teachers\_from\_db()

    assert(len(teachers) > 0)

unit\_test()

**Under ViewUpdateExams.py section**

import sqlite3

class Subject:

    def \_\_init\_\_(self,uuid,CA1,SA1,CA2,SA2):

        # Object Attributes

        self.uuid = uuid

        self.CA1 = CA1

        self.SA1 = SA1

        self.CA2 = CA2

        self.SA2 = SA2

def update\_multiple\_columns(StudentUUID, Subject, CA1, SA1, CA2, SA2):

    # Function has been adapted from https://pynative.com/python-sqlite-update-table/

    # Function is used to update student's exam marks in the database

    sqliteConnection = sqlite3.connect('students.db3')

    # Connects to database ‘students.db3’ and stores it in variable sqliteConnection.

    cursor = sqliteConnection.cursor()

    # Connects to the database 'students.db3' and allows user to data manipulate anything

    # inside the database through the temporary workstation, stores it in a variable called cursor.

    sqlite\_update\_query = "Update " + Subject + " set CA1 = ?, SA1 = ?, CA2 = ?, SA2 = ? where StudentUUID = ?"

    # Is a string that stores the Subject positional argument variable into the string.

    columnValues = (CA1, SA1, CA2, SA2, StudentUUID)

    # Applies the positional argument variable data inside this tuple columnValues

    cursor.execute(sqlite\_update\_query, columnValues)

    # The values stored in the columnValues binds it to the string sqlite\_update\_query.

    # The execute is then pulling data from the database under potential subject tables

    # such as English, Math, Chinese, Science that have the columns CA1,SA1,CA2,SA2 and StudentUUID

    # which then promptly updates the marks of these 4 exams under the StudentUUID listed in the variable.

    sqliteConnection.commit()

    # Confirms any modifications made in the update query into the students.db3 database

    cursor.close()

    # Closes the connection to the database

def unit\_test():

    update\_multiple\_columns("S000015", "Math", 42, 24, 98, 88)

unit\_test()

**Under main.py section**

from tkinter import \*

from tkinter import ttk

import sqlite3

from ReadStudentAccounts import \*

from ReadTeacherAccounts import \*

from ViewUpdateExams import \*

list\_of\_subjects = ["English", "Math", "Chinese", "Science"]

list\_of\_exams = ["CA1", "SA1", "CA2", "SA2"]

# Preloaded sets of lists in the code to allow easy access for functions later in the code.

students = get\_all\_students\_from\_db()

teachers = get\_all\_teachers\_from\_db()

# Preloaded variables that are lists that contain all the information about students and teachers accounts

# Allow easy access for functions later in the code.

# Picks function, login verification, based on radio button clicked

def log\_student\_or\_teacher():

    val = var.get()

    # Obtains the IntVar() of var and will store the integer number into val

    print(val)

    if val == 1:

        student\_login\_verify()

    # If val number is 1, indicating that the user clicked on the radiobutton with Student labelled on it,

    # it will bring the function to the student login verification

    elif val == 2:

        teacher\_login\_verify()

    # If val number is 2, indicating that the user clicked on the radiobutton with Teacher labelled on it,

    # it will bring the function to the teacher login verification

# Picks function based on what is inputted, subject or student UUID

def subject\_or\_student\_marks():

    global subject\_or\_student # Makes variable subject\_or\_student from local to global

    subject\_or\_student = subject\_or\_mark.get()

    # Gets the StringVar() subject\_or\_mark and turns it into a variable string subject\_or\_student

    if any(x in subject\_or\_student for x in list\_of\_subjects):

        subject\_exam\_results()

    # If statement to check if the variable string subject\_or\_student is in the list list\_of\_subjects,

    # if it is in the list list\_of\_subjects, it runs the function subject\_exam\_results()

    for s in students:

    # s will be a variable that iterates over the tuple elements that contain the Student class

    # object attributes in the students list through a for loop

        if subject\_or\_student == s.uuid:

            get\_student\_marks()

        # Runs an if statement to check if the string variable subject\_or\_student is the same as the variable

        # with object attribute s.uuid, if it is, it runs the function get\_student\_marks()

# Get all marks of one student stscreen

def get\_student\_marks\_stscreen(): # Function has been adapted from https://pynative.com/python-sqlite-select-from-table/

    sqliteConnection = sqlite3.connect('students.db3')

    # Connects to database ‘students.db3’ and stores it in variable sqliteConnection.

    cursor = sqliteConnection.cursor()

    # Connects to the database 'students.db3' and allows user to data manipulate anything

    # inside the database through the temporary workstation, stores it in a variable called cursor.

    results = []

    # results is a list

    for x in list\_of\_subjects:

    # Runs a for loop and iterates over the list, list\_of\_subjects under the variable x

    # which contains the elements [English, Math, Chinese, Science]

        query = "SELECT CA1, SA1, CA2, SA2 FROM " + x + " where StudentUUID = ?"

        # Selects the table in order of the list\_of\_subjects, which then selects the exam results of the StudentUUID acquired

        cursor.execute(query, (student\_uuid,))

        # Executes variable string name sqlite\_select\_query through the cursor

        # with the variable student\_uuid, who is the user that logged in currently

        results = cursor.fetchall()

        # Fetches all 4 subject data and stores them inside the results list.

        for result in results:

            tree.insert("", END, values=([x], \*result))

        # Runs a for loop and inserts exam results into the tree. Values are the headings, and x contains each of the subject names.

        # Result is the iterated tuples in the results list that will store all the data into each of these headings.

# Get all marks of one student teacherscreen

def get\_student\_marks(): # Function has been adapted from https://pynative.com/python-sqlite-select-from-table/

    sqliteConnection = sqlite3.connect('students.db3')

    # Connects to database ‘students.db3’ and stores it in variable sqliteConnection.

    cursor = sqliteConnection.cursor()

    # Connects to the database 'students.db3' and allows user to data manipulate anything

    # inside the database through the temporary workstation, stores it in a variable called cursor.

    results = []

    # results is a list

    for x in list\_of\_subjects:

    # Runs a for loop and iterates over the list, list\_of\_subjects under the variable x

    # which contains the elements [English, Math, Chinese, Science]

        query = "SELECT CA1, SA1, CA2, SA2 FROM " + x + " where StudentUUID = ?"

        # Selects the table in order of the list\_of\_subjects, which then selects the exam results of the StudentUUID acquired

        cursor.execute(query, (subject\_or\_student,))

        # Executes variable string name sqlite\_select\_query through the cursor with the variable subject\_or\_student

        results = cursor.fetchall()

        # Fetches all 4 subject data and stores them inside the results list.

        for result in results:

            tree2.insert("", END, values=([x], \*result))

        # Runs a for loop and inserts exam results into the tree. Values are the headings, and x contains each of the subject names.

        # Result is the iterated tuples in the results list that will store all the data into each of these headings.

# Gets all student marks in one subject

def subject\_exam\_results(): # Function has been adapted from https://pynative.com/python-sqlite-select-from-table/

    sqliteConnection = sqlite3.connect('students.db3')

    # Connects to database with the file name ‘students.db3’ and stores it in a variable called sqliteConnection.

    cursor = sqliteConnection.cursor()

    # Connects to the database 'students.db3' and allows user to data manipulate anything

    # inside the database through the temporary workstation, stores it in a variable called cursor.

    sqlite\_select\_query = "SELECT \* from " + subject\_or\_student

    # Selects all of the data from the table name based on what is in the string variable subject\_or\_student.

    cursor.execute(sqlite\_select\_query)

    # Executes variable string name sqlite\_select\_query through the cursor

    records = cursor.fetchall()

    # Fetches all student exam results under the table of the string variable subject\_or\_student

    for row in records:

        tree2.insert("", END, values=row)

    # Runs a for loop and inserts exam results into the tree through the variable row.

    # Values are the headings, and row contains each of the student exam results.

    # row is the iterated tuples in the list records.

    sqliteConnection.close()

    # Closes connection to sqlite

# Used to check username\_verify and password\_verify if they are a student

# and identify if the login credentials exist in the database.

def student\_login\_verify(): # Function has been adapted from https://www.simplifiedpython.net/python-gui-login/

    global student\_name # Makes variable student\_name from local to global

    global student\_uuid # Makes variable student\_uuid from local to global

    username1 = username\_verify.get()

    # Obtains the StringVar() of username\_verify and will store the string into the variable username1.

    password1 = password\_verify.get()

    # Obtains the StringVar() of password\_verify and will store the string into the variable password1

    student = None

    # Creates a variable called student that currently stores nothing

    for s in students:

    # s will be a variable that iterates over the tuple elements

    # that contain the Student class object attributes in the students list through a for loop

        if username1 == s.uuid:

        # When s iterates over the whole students list, it will check if username1 is the same as the first element

        # of the tuples in students list, which holds the uuids of the students as the Student class object attribute uuid.

            student = s

            # If there is a matching uuid, it will store that tuple in student.

    if not student == None:

    # If student holds any tuples, it will continue

        if password1 == student.password:

        # If the password1 is the same as the object attribute of Student class, student.password, it will continue

            student\_name = student.name

            # Store the student.name as the variable student\_name

            student\_uuid = student.uuid

            # Store the student.uuid as the variable student\_uuid

            student\_screen()

            # Bring the user to the function student\_screen()

        else:

            password\_not\_recognised()

    # If password1 does not match with student.password,

    # it will bring the user to the function password\_not\_recognised()

    else:

        user\_not\_found()

    # If student is still None, indicating the username1 is not in the StudentAccounts UUID database,

    # it will run the function user\_not\_found()

# Used to check username\_verify and password\_verify if they are a teacher

# and identify if the login credentials exist in the database.

def teacher\_login\_verify(): # Function has been adapted from https://www.simplifiedpython.net/python-gui-login/

    global teacher\_name # Makes variable teacher\_name from local to global

    username1 = username\_verify.get()

    # Obtains the StringVar() of username\_verify and will store the string into the variable username1.

    password1 = password\_verify.get()

    # Obtains the StringVar() of password\_verify and will store the string into the variable password1

    teacher = None

    # Creates a variable called teacher that currently stores nothing

    for t in teachers:

    # t will be a variable that iterates over the tuple elements

    # that contain the Teacher class object attributes in the teachers list through a for loop

        if username1 == t.uuid:

        # When t iterates over the whole teachers list, it will check if username1 is the same as the first element

        # of the tuples in teachers list, which holds the uuids of the teachers as the Teacher class object attribute uuid.

            teacher = t

            # If there is a matching uuid, it will store that tuple in teacher.

    if not teacher == None:

    # If teacher holds any tuples, it will continue

        if password1 == teacher.password:

        # If the password1 is the same as the object attribute of Teacher class, teacher.password, it will continue

            teacher\_name = teacher.name

            # Store the teacher.name as the variable teacher\_name

            teacher\_screen()

            # Bring the user to the function teacher\_screen()

        else:

            password\_not\_recognised()

    # If password1 does not match with student.password,

    # it will bring the user to the function password\_not\_recognised()

    else:

        user\_not\_found()

    # If teacher is still None, indicating the username1 is not in the TeacherAccounts UUID database,

    # it will run the function user\_not\_found()

# Update all marks of one student in one subject into the database.

def update\_marks():

    uuid = UUID\_Update.get()

    # Obtains the StringVar() of UUID\_Update and will store the string into the variable uuid.

    subject = Subject\_Update.get()

    # Obtains the StringVar() of Subject\_Update and will store the string into the variable subject.

    student = None

    # Variable student contains None

    ca1 = CA1\_marks.get()

    # Obtains the IntVar() of CA1\_marks and store the integer into the variable ca1.

    sa1 = SA1\_marks.get()

    # Obtains the IntVar() of SA1\_marks and store the integer into the variable sa1.

    ca2 = CA2\_marks.get()

    # Obtains the IntVar() of CA2\_marks and store the integer into the variable ca2.

    sa2 = SA2\_marks.get()

    # Obtains the IntVar() of SA2\_marks and store the integer into the variable ca2.

    if ca1 and sa1 and ca2 and sa2 in range(0,101):

    # If all 4 of these int var in range of 1-100, it will continue

        if subject in list\_of\_subjects:

        # If string variable subject in list list\_of\_subjects, it will continue.

            for s in students:

            # s will be a variable that iterates over the tuple elements

            # that contain the Student class object attributes in the students list through a for loop

                if uuid == s.uuid:

                    student = s

                # When s iterates over the whole students list, it will check if uuid is the same

                # as the first element of the tuples in students list, which holds the uuids of the

                # students as the Student class object attribute uuid. If there is a matching uuid,

                # it will store that tuple in student.

                    update\_multiple\_columns(uuid, subject, ca1, sa1, ca2, sa2)

                    # Stores all the variables into update\_multiple\_columns()

                    update\_successful()

            if not student == None:

            # If student holds any tuples, it will continue.

                return # Breaks out of function

            else:

                user\_not\_found()

            # If student is still None,

            # indicating the uuid is not in the StudentAccounts UUID database,

            # it will run the function user\_not\_found()

        else:

             subject\_not\_found()

        # Bring the user to the function subject\_not\_found() if string variable subject is not in list subject

    else:

        marks\_not\_found()

    # Bring the user to the function marks\_not\_found() if its not in range of 1-100

#set conditions for update, marks can only be from 1-100, uuid and subject must be a student and subject from the database

# Designing popup for login invalid password

def password\_not\_recognised(): # Function has been imported from https://www.simplifiedpython.net/python-gui-login/

    global password\_not\_recog\_screen # Makes variable password\_not\_recog\_screen from local to global

    password\_not\_recog\_screen = Toplevel(login\_screen)

    # Places password\_not\_recog\_screen on top of login screen

    password\_not\_recog\_screen.title("")

    # Empty title

    password\_not\_recog\_screen.geometry("150x100")

    # 150 wide and 100 pixels high

    Label(password\_not\_recog\_screen, text="Invalid Password ").pack()

    # Packed label set to password\_not\_recog\_screen, that displays “Invalid Password”

    Button(password\_not\_recog\_screen, text="OK", command = delete\_password\_not\_recognised).pack()

    # Packed button set to password\_not\_recog\_screen, that runs the function delete\_password\_not\_recognised()

# Designing popup for user not found

def user\_not\_found(): # Function has been imported from https://www.simplifiedpython.net/python-gui-login/

    global user\_not\_found\_screen # Makes variable user\_not\_found\_screen from local to global

    user\_not\_found\_screen = Toplevel(login\_screen)

    # Places user\_not\_found\_screen on top of login screen

    user\_not\_found\_screen.title("")

    # Empty title

    user\_not\_found\_screen.geometry("150x100")

    # 150 wide and 100 pixels high

    Label(user\_not\_found\_screen, text = "User Not Found").pack()

    # Packed label text set to user\_not\_found\_screen, that displays “User Not Found”

    Button(user\_not\_found\_screen, text = "OK", command = delete\_user\_not\_found\_screen).pack()

    # Packed button set to user\_not\_found\_screen that runs the function delete\_user\_not\_found()

# Designing popup for subject not found

def subject\_not\_found():

    global subject\_not\_found\_screen

    subject\_not\_found\_screen = Toplevel(updatemarksscreen)

    # Places subject\_not\_found\_screen on top of updatemarksscreen

    subject\_not\_found\_screen.title("")

    # Empty title

    subject\_not\_found\_screen.geometry("150x100")

    # 150 wide and 100 pixels high

    Label(subject\_not\_found\_screen, text = "Unknown Subject").pack()

    # Packed label text set to subject\_not\_found\_screen, that displays “Unknown Subject”

    Button(subject\_not\_found\_screen, text = "OK", command = delete\_subject\_not\_found\_screen).pack()

    # Packed button set to subject\_not\_found\_screen that runs the function delete\_subject\_not\_found\_screen()

# Designing popup for subject not found

def marks\_not\_found():

    global marks\_not\_found\_screen

    marks\_not\_found\_screen = Toplevel(updatemarksscreen)

    # Places marks\_not\_found\_screen on top of updatemarksscreen

    marks\_not\_found\_screen.title("")

    # Empty title

    marks\_not\_found\_screen.geometry("150x100")

    # 150 wide and 100 pixels high

    Label(marks\_not\_found\_screen, text = "Marks not within 1-100").pack()

    # Packed label text set to marks\_not\_found\_screen, that displays “Marks not within 1-100” on marks\_not\_found\_screen

    Button(marks\_not\_found\_screen, text = "OK", command = delete\_marks\_not\_found\_screen).pack()

    # Packed button set to marks\_not\_found\_screen that runs the function delete\_marks\_not\_found\_screen()

# Designing popup for marks being successfully updated

def update\_successful():

    global update\_successful\_screen

    update\_successful\_screen = Toplevel(updatemarksscreen)

    # Places update\_successful\_screen on top of updatemarksscreen

    update\_successful\_screen.title("")

    # Empty title

    update\_successful\_screen.geometry("150x100")

    # 150 wide and 100 pixels high

    Label(update\_successful\_screen, text = "Update Successful").pack()

    # Packed label text set to update\_successful\_screen, that displays "Update Successful" on update\_successful\_screen

    Button(update\_successful\_screen, text = "OK", command = delete\_update\_successful\_screen).pack()

    # Packed button set to update\_successful\_screen that runs the function delete\_update\_successful\_screen

# Designing Log out Button

def log\_out():

    main\_screen.update()

    main\_screen.deiconify()

    login\_screen.destroy()

# Designing clear all TreeView Button

def clear\_Tree():

    for row in tree.get\_children():

        tree.delete(row)

    # Runs a for loop and it deletes all rows of displayed tree data on the stscreen.

def clear\_Tree2():

    for row in tree2.get\_children():

        tree2.delete(row)

    # Runs a for loop and it deletes all rows of displayed tree data on the teacherscreen.

# Deleting popups

def delete\_password\_not\_recognised(): # Function has been imported from https://www.simplifiedpython.net/python-gui-login/

    password\_not\_recog\_screen.destroy()

    # Destroys the window password\_not\_recog\_screen

def delete\_user\_not\_found\_screen(): # Function has been imported from https://www.simplifiedpython.net/python-gui-login/

    user\_not\_found\_screen.destroy()

    # Destroys the window user\_not\_found\_screen

def delete\_subject\_not\_found\_screen():

    subject\_not\_found\_screen.destroy()

    # Destroys the window subject\_not\_found\_screen screen

def delete\_marks\_not\_found\_screen():

    marks\_not\_found\_screen.destroy()

    # Destroys the window marks\_not\_found\_screen screen

def delete\_update\_successful\_screen():

    update\_successful\_screen.destroy()

    # Destroys the window update\_successful\_screen screen

# Designing Main(first) window

def main\_account\_screen(): # Function has been adapted from https://www.simplifiedpython.net/python-gui-login/

    global main\_screen # Sets variable main\_screen from local to global

    global var # Sets variable var from local to global

    main\_screen = Tk()

    # Parent screen stored in the variable main\_screen

    var = IntVar()

    # Changes var to integer data

    var.set(1)

    # Sets var to Student radiobutton initially

    main\_screen.geometry("350x200")

    # 350 wide and 200 pixels high

    main\_screen.title("Account Login")

    # Application title “Account Login”

    Label(main\_screen, text="Select Your Choice", bg="blue", width="300", height="2", font=("Calibri", 13)).pack()

    # Packed label will display on the main\_screen, with the text “Select your choice”, with a font of Calibri size 13,

    # in a blue background at a width of 300 and height of 2 characters.

    Radiobutton(main\_screen, text="Student", variable = var, value = 1).pack(anchor = NW)

    # Packed radiobutton is set to main\_screen and the text displayed for the first option is “Student”,

    # the variable var is put into a IntVar() function that stores this integer value

    # and var.set(1) auto sets the choice initially to 1, is bound to NW which is north west.

    Radiobutton(main\_screen, text="Teacher", variable = var, value = 2).pack(anchor = NW)

    # Packed radiobutton’s second option will be “Teacher”, and when is clicked will set var value to 2,

    # is bound to NW which is north west.

    Button(main\_screen, text = "Login", height = "2", width = "30", command = login).pack()

    # Packed button is set to main\_screen, “Login” as its text, a width of 30 and height of 2 characters and a command that is set to login().

    # login() is a function that will bring the user from the main\_screen to the login\_screen as depicted.

    main\_screen.mainloop()

    # Loops tkinter event

# Designing window for login

def login(): # Function has been adapted from https://www.simplifiedpython.net/python-gui-login/

    global login\_screen # Sets variable login\_screen from local to global

    main\_screen.withdraw()

    # Hides the main screen without destroying it.

    login\_screen = Toplevel(main\_screen)

    # Places the login\_screen on top of the main\_screen

    login\_screen.title("Login")

    # Application title “Login”

    login\_screen.geometry("300x250")

    # 350 wide and 200 pixels high

    Label(login\_screen, text="Please enter details below to login").pack()

    # Packed label will display on the login\_screen, with the text "Please enter details below to login”

    global username\_verify

    # sets username\_verify from local to global

    global password\_verify

    # sets username\_verify from local to global

    username\_verify = StringVar()

    # Changes username\_verify to string data

    password\_verify = StringVar()

    # Changes password\_verify to string data

    Label(login\_screen, text="Username \* ").pack()

    # Packed label will display on the login\_screen, with the text "Username”

    Entry(login\_screen, textvariable = username\_verify).pack()

    # Packed entry that will store anything user types as textvariable username\_verify

    Label(login\_screen, text="Password \* ").pack()

    # Packed label will display on the login\_screen, with the text "Password”

    Entry(login\_screen, textvariable = password\_verify, show= "\*").pack()

    # Packed entry that will store anything user types as textvariable password\_verify

    Button(login\_screen, text="Login", width=10, height=1, command = log\_student\_or\_teacher).pack()

    # Packed button set to login\_screen that will do a login verification based on function log\_student\_or\_teacher()

# Designing Student Screen

def student\_screen():

    global stscreen # Sets variable stscreen from local to global

    global tree # Sets variable tree from local to global

    login\_screen.withdraw()

    # Hides login\_screen without destroying it

    stscreen = Toplevel(login\_screen)

    # Places stscreen on top of login screen

    stscreen.geometry("600x400")

    # 600 wide and 400 pixels high

    stscreen.title("Welcome, " + student\_name)

    # Creates a title, an example of what it displays could be “Welcome, Student1” on stscreen

    # if the logged in credentials of S000001 was keyed in the login section correctly

    Button(stscreen, text="Log Out", padx=3, pady=3, command=log\_out, fg="white", bg="red").pack(anchor = NE)

    # Packed button set to stscreen that runs the function log\_out() that is 3 pixels wide and tall,

    # displaying the text “Log Out” in white with a red background that is anchored on the north-east part of the stscreen

    tree = ttk.Treeview(stscreen, column=("c1", "c2", "c3", "c4", "c5"), show='headings')

    # Stores a Treeview in variable tree with column ids of c1, c2, c3, c4, c5 and column heading to display text as string ‘headings’

    tree.column("#1", minwidth=0, width=70, anchor=CENTER, stretch=NO)

    # Tree column 70 pixels wide, centralises text and no stretch

    tree.heading("#1", text = student\_name)

    # Displays the heading underneath the column to be student’s name, e.g account logged in is S000001,

    # the displayed text will be Student1.

    for (i,subject) in enumerate(list\_of\_exams):

    # for loop has two variables i and subject that will iterate over the list list\_of\_exams,

    # that contains [English, Math, Chinese, Science]. i will be an integer to display

    # the number of iterations through the loop and the subject will be a string that will display

    # what subject is being iterated over currently.

        tree.column("#"+str(i+2), minwidth=0, width=40, anchor=CENTER, stretch=NO)

        # i starts off at an iteration of 0 loops, which will produce the column id #2, having a width of 40 pixels

        # this time, and increases by one after each loop to produce #3, #4 and #5

        tree.heading("#"+str(i+2), text = subject)

        # Displays any text in the column heading of the variable subject, which can all be referenced in the function

        # get\_students\_marks\_st\_screen(), which contains all the text inserted into each of these column headings.

    tree.pack()

    # Packs the tree and displays it on the student\_screen

    Button(stscreen, text="Display Marks", command=get\_student\_marks\_stscreen).pack(pady=10)

    # Packed button set to stscreen that has the text “Display Marks”

    # that runs the function get\_student\_marks\_stscreen() when clicked

    Button(stscreen, text="Clear data", command = clear\_Tree).pack(pady=10)

    # Packed button set to stscreen that has the text “Clear data” that runs the function clear\_Tree() when clicked

# Designing Teacher Screen

def teacher\_screen():

    global teacherscreen # Sets variable teacherscreen from local to global

    global subject\_or\_mark # Sets variable subject\_or\_mark from local to global

    global tree2 # Sets variable tree2 from local to global

    login\_screen.withdraw()

    # Hides the login\_screen without destroying it.

    teacherscreen = Toplevel(login\_screen)

    # Places teacherscreen on top of login screen

    teacherscreen.geometry("400x600")

    # 400 wide and 600 pixels high

    teacherscreen.title("Welcome, " + teacher\_name)

    # Creates a title, an example of what it displays could be “Welcome, Teacher1”

    # on teacherscreen if the logged in credentials of T000001 was keyed in the login section correctly

    subject\_or\_mark = StringVar()

    # Changes subject\_or\_mark to string data

    Button(teacherscreen, text="Log Out", padx=3, pady=3, command=log\_out, fg="white", bg="red").pack(anchor = NE)

    # Packed button set to stscreen that runs the function log\_out() that is 3 pixels wide and tall,

    # displaying the text “Log Out” in white with a red background that is anchored on the north-east part of the teacherscreen

    Label(teacherscreen, text="View Subject or Student \* ").pack()

    # Packed label that displays the text “View Subject or Student” on teacherscreen

    Entry(teacherscreen, textvariable = subject\_or\_mark).pack()

    # Packed entry that will store anything user types as textvariable subject\_or\_mark

    tree2 = ttk.Treeview(teacherscreen, column=("c1", "c2", "c3", "c4", "c5"), show='headings')

    # Stores a Treeview in variable tree2 with column ids of c1, c2, c3, c4, c5 and column heading to display text as string ‘headings’

    tree2.column("#1", minwidth=0, width=70, anchor=CENTER, stretch=NO)

    # Tree2 column 70 pixels wide, centralises text and no stretch

    tree2.heading("#1", text= "")

    # Tree2 heading does not display any text

    for (i,subject) in enumerate(list\_of\_exams):

    # for loop has two variables i, subject that will iterate over the list list\_of\_exams, that contains [English, Math, Chinese, Science].

    # i will be an integer to display the number of iterations through the loop

    # and the subject will be a string that will display what subject is being iterated over currently.

        tree2.column("#"+str(i+2), minwidth=0, width=40, anchor=CENTER, stretch=NO)

        # for loop has two variables i, subject that will iterate over the list list\_of\_exams, that contains [English, Math, Chinese, Science].

        # i will be an integer to display the number of iterations through the loop and the subject will be a string

        # that will display what subject is being iterated over currently.

        tree2.heading("#"+str(i+2), text = subject)

        # Displays any text in the column heading of the variable subject, which can all be referenced in the function get\_students\_marks() or subject\_exam\_results(),

        # which contains all the text inserted into each of these column headings.

    tree2.pack()

    # Packs the tree and displays it on the teacher\_screen

    Button(teacherscreen, text="Display data", command=subject\_or\_student\_marks).pack(pady=10)

    # Packed button on teacherscreen, has a size of 10 pixels high, displays text “Display Data” which runs the function subject\_or\_student\_marks()

    Button(teacherscreen, text="Clear data", command = clear\_Tree2).pack(pady=10)

    # Packed button on teacherscreen, has a size of 10 pixels high, displays text “Clear data” which runs the function clear\_Tree2()

    Button(teacherscreen, text="Update marks", command = update\_marks\_screen).pack(pady=10)

    # Packed button on teacherscreen, has a size of 10 pixels high, displays text “Update marks” which runs the function update\_marks\_screen()

# Obtain entries to update marks in database

def update\_marks\_screen():

    global updatemarksscreen # Sets variable updatemarksscreen from local to global

    global UUID\_Update # Sets variable UUID\_Update from local to global

    global Subject\_Update # Sets variable Subject\_Update from local to global

    global CA1\_marks # Sets variable CA1\_marks from local to global

    global SA1\_marks # Sets variable SA1\_marks from local to global

    global CA2\_marks # Sets variable CA2\_marks from local to global

    global SA2\_marks # Sets variable SA2\_marks from local to global

    updatemarksscreen = Toplevel(teacherscreen)

    # Places the updatemarksscreen on top of the teacherscreen

    updatemarksscreen.geometry("400x300")

    # 400 wide and 300 pixels high

    updatemarksscreen.title("Update marks here")

    # Application title “Update marks here”

    UUID\_Update = StringVar()

    # Changes UUID\_Update to string data

    Subject\_Update = StringVar()

    # Changes Subject\_Update to string data

    CA1\_marks = IntVar()

    # Changes CA1\_marks to integer data

    SA1\_marks = IntVar()

    # Changes SA1\_marks to integer data

    CA2\_marks = IntVar()

    # Changes CA2\_marks to integer data

    SA2\_marks = IntVar()

    # Changes SA2\_marks to integer data

    Label(updatemarksscreen, text="Student ID \* ").pack()

    # Packed label that displays the text "Student ID” on updatemarksscreen

    Entry(updatemarksscreen, textvariable = UUID\_Update).pack()

    # Packed entry that will store anything user types as textvariable UUID\_Update

    Label(updatemarksscreen, text="Subject \* ").pack()

    # Packed label that displays the text "Subject” on updatemarksscreen

    Entry(updatemarksscreen, textvariable = Subject\_Update).pack()

    # Packed entry that will store anything user types as textvariable Subject\_Update

    Label(updatemarksscreen, text="CA1 \* ").pack()

    # Packed label that displays the text "CA1” on updatemarksscreen

    Entry(updatemarksscreen, textvariable = CA1\_marks).pack()

    # Packed entry that will store anything user types as textvariable CA1\_marks

    Label(updatemarksscreen, text="SA1 \* ").pack()

    # Packed label that displays the text "SA1” on updatemarksscreen

    Entry(updatemarksscreen, textvariable = SA1\_marks).pack()

    # Packed entry that will store anything user types as textvariable SA1\_marks

    Label(updatemarksscreen, text="CA2 \* ").pack()

    # Packed label that displays the text "CA2” on updatemarksscreen

    Entry(updatemarksscreen, textvariable = CA2\_marks).pack()

    # Packed entry that will store anything user types as textvariable CA2\_marks

    Label(updatemarksscreen, text="SA2 \* ").pack()

    # Packed label that displays the text "SA2” on updatemarksscreen

    Entry(updatemarksscreen, textvariable = SA2\_marks).pack()

    # Packed entry that will store anything user types as textvariable SA2\_marks

    Button(updatemarksscreen, text="Update Marks", width= 10, height= 1, command = update\_marks).pack()

    # Packed button set to updatemarksscreen that will update marks based on function update\_marks()

main\_account\_screen()

# Loads function main\_account\_screen

**Section 5: Testing to inform development.**

For these two functions, it can be identified that they are working as intended if the assert does not send back any AssertionError. This indicates that the data from the database is going into the lists teachers and students as intended.

**ReadStudentAccounts.py**

def unit\_test():

    students = get\_all\_students\_from\_db()

    assert(len(students) > 0)

unit\_test()

**ReadTeacherAccounts.py**

def unit\_test():

    teachers = get\_all\_teachers\_from\_db()

    assert(len(teachers) > 0)

unit\_test()

**ViewUpdateExams.py**

For the update\_multiple\_columns(StudentUUID, Subject, CA1, SA1, CA2, SA2) function, a more specific unit test is needed to prove it updates data into the database as follows:

def unit\_test():

    update\_multiple\_columns("S000015", "Math", 42, 24, 98, 88)

Output:



**Student login**

|  |  |  |
| --- | --- | --- |
| Username | Password | Result |
| S000021  S000000  T000001 | aaa  aaa  aaa |  |
| S000001  S000001  S000001 | AAA  qwerty  111 | A screenshot of a computer  Description automatically generated |

**Teacher login**

|  |  |  |
| --- | --- | --- |
| Username | Password | Result |
| T000021  T000000  S000001 | aaa  aaa  aaa |  |
| T000001  T000001  T000001 | AAA  qwerty  111 | A screenshot of a computer  Description automatically generated |

**Teacher Screen, Update marks screen**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Subject | CA1 | SA1 | CA2 | SA2 | Result |
| S000001  S000001  S000001  S000001 | Math  Math  Math  Math | 0  -1  101  A | 0  -1  101  A | 0  -1  101  A | 0  -1  101  A |  |
| S000021  T000001 | Math Math | 15  15 | 15  15 | 15  15 | 15  15 |  |
| S000001  S000001 | A  Chemistry | 15  15 | 15  15 | 15  15 | 15  15 |  |

**Section 6: Testing to inform evaluation:**

**Student login page:**

**A screenshot of a login screen

Description automatically generated with medium confidence**

**Student Screen:**

A screenshot of a computer

Description automatically generated

**Student Screen, Display marks:**

A screenshot of a computer

Description automatically generated

**Student Screen, Clear data:**

A screenshot of a computer

Description automatically generated

**Log out:**

A screenshot of a login screen

Description automatically generated with medium confidence

**Teacher login page:**

**A screenshot of a login screen

Description automatically generated with medium confidence**

**Teacher Screen:**

A screenshot of a computer

Description automatically generated with medium confidence

**Teacher Screen, Display student data:**

**A screenshot of a computer

Description automatically generated**

**Teacher Screen, Clear data:**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Teacher Screen, Display subject data:**

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer screen

Description automatically generated with low confidenceA picture containing text, screenshot, number, font

Description automatically generated

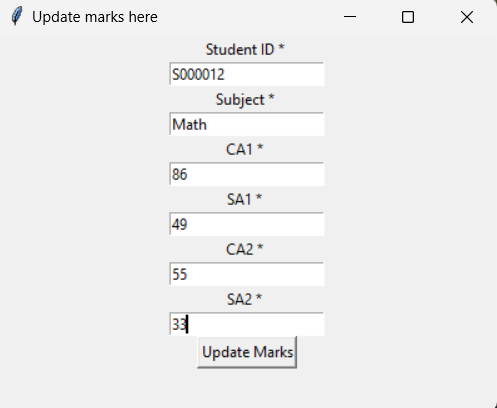
Database for comparison

**Teacher Screen, Log out:**

**A screenshot of a login screen

Description automatically generated with medium confidence**

**Update marks screen:**

**** **A screenshot of a computer

Description automatically generated**

**A screenshot of a table

Description automatically generated**

Database for comparison of S000012

**Section 7: Evaluation of Solution**

For most of the analysis report’s ideas has been achieved, such as a UUID for students and teachers, a user status for them, their names are displayed on the title of their user interface when logged in. They can login with a UUID and password, access a radiobutton that can bring them to the login page of either the student verification or the teacher verification, the student can also view their own scores in their separate screen. The teacher is able to view a specific student’s scores, or a subject’s list of student scores. The teacher is also able to update marks accordingly. In both screens, they are able to log out. The teacher is able to retrieve and display the details of a particular student, i.e their exam scores. With two reports, being either the student or the subject Rob is able to identify from the entry box in teacher screen to display their data.

However, there is no breakdown of the course information and the assessments in the course, plus students are unable to see a combined total for their course. There is also no admin client to insert new data and delete existing data in the GUI.

However, all of the minimal requirements have been achieved and most of the needs of the stakeholders has been achieved.

Other interesting functionalities could include being able to host classes for teachers and students, organising a timetable for each student and teacher to view.

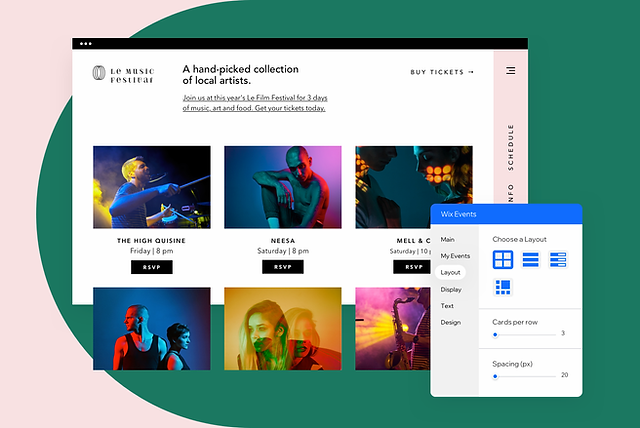
Some functionalities that should be added are encryption of passwords, it is very important for maintaining security in any database, not only that, but the passwords also have to be salted too before encryption because hackers can easily access rainbow tables to easily decrypt basic encrypted passwords. Rainbow tables tends to be a database designed to cache the outputted cryptographic hash functions something similar to this, hence salting, which is just adding a string or value to the password before encrypting the password, can ensure rainbow tables are less effective A diagram of a computer program

Description automatically generated

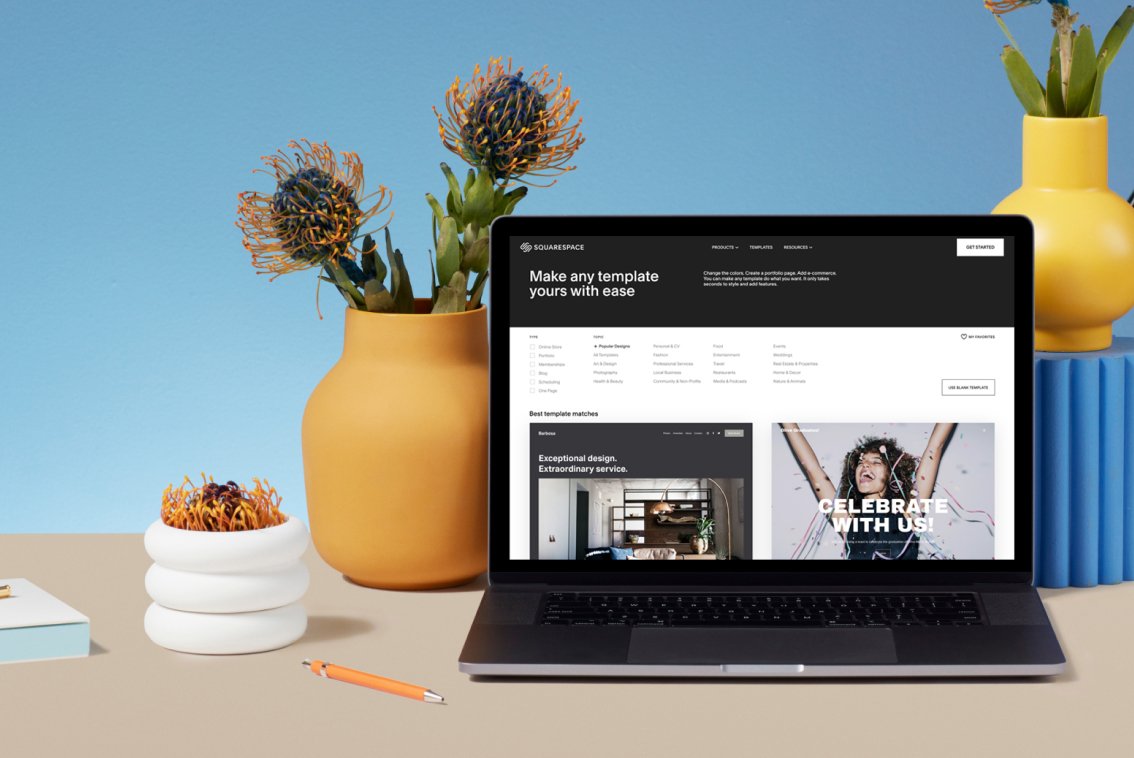
(Rainbow Table, 2023).

Another security improvement could be doing the two-step verification method similar to how Kaplan implements it, after you enter your email and password, they will ask you to verify your email by sending a verification code to your email.

Another improvement could be using a web interface versus a user interface, for upscaling purposes that allows more users to access the application at once. Not only that it significantly requires less storage for the OS versus using a website. A graphic user interface tends to also be very rigid, and there are many websites nowadays that provide templates to create websites under they’re URL, such as wix and squarespace.



(Sutherland, 2022)



(Mitchell, 2022)

Another improvement to the database is using google cloud sql versus sqlite, it creates easy scalability. Though using google cloud tends to have better security than most databases, as it is owned by google. This can result in less risk in data breaches or leaks. Not only that, but it is also easier to use since there is a wide range of tools to make the management of data in the database easier. However, there are some downsides to this. Google cloud would require the user to be connected to the world wide web versus sqlite where it can be offline, SQlite also tends to be more lightweight versus google cloud.

**References**

Mitchell, N 2022, How to Design a Website for Your Brand, image, Squarespace, viewed 8 July 2023, <<https://www.squarespace.com/blog/guide-to-website-design>>.

Python Functions, w3Schools, viewed 4 July 2023, <<https://www.w3schools.com/python/python_functions.asp>>.

Python lists, w3Schools, viewed 4 July 2023, <<https://www.w3schools.com/python/python_lists.asp>>.

Python List append() Method, w3Schools, viewed 4 July 2023, <<https://www.w3schools.com/python/ref_list_append.asp>>.

Python for loops, w3Schools, viewed 4 July 2023, <<https://www.w3schools.com/python/python_for_loops.asp>>.

Python tuples, w3Schools, viewed 4 July 2023, <<https://www.w3schools.com/python/python_tuples.asp>>.

Python return Keyword, w3Schools, viewed 4 July 2023, <<https://www.w3schools.com/python/ref_keyword_return.asp>>.

Python – Global Variables, w3Schools, viewed 4 July 2023, <<https://www.w3schools.com/python/python_variables_global.asp>>.

2023, Rainbow Table, Wikipedia, viewed 6July 2023, <<https://en.wikipedia.org/wiki/Rainbow_table>>.

Sutherland, R 2022, How to make a website with Wiz, image, Wix, viewed 8 Jule 2023, <<https://www.tomsguide.com/how-to/how-to-build-a-website-on-wix>>.

[sqlite3](https://docs.python.org/3/library/sqlite3.html#module-sqlite3) — DB-API 2.0 interface for SQLite databases, Python, viewed 4 July 2023, <<https://docs.python.org/3/library/sqlite3.html>>.

Tkinter – python interface to Tcl/Tk, Python, viewed 4 July 2023, <<https://docs.python.org/3/library/tkinter.html>>.

Treeview, TkDocs, viewed 4 July 2023, <<https://tkdocs.com/tutorial/tree.html>>.

Variable classes in Tkinter (2021), python-course, viewed 4 July 2023 <<https://python-course.eu/tkinter/variable-classes-in-tkinter.php>>.

What is a constructor in Python?, Pythonbasics, viewed 4 July 2023, <<https://pythonbasics.org/constructor/>>.

**Appendix**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated