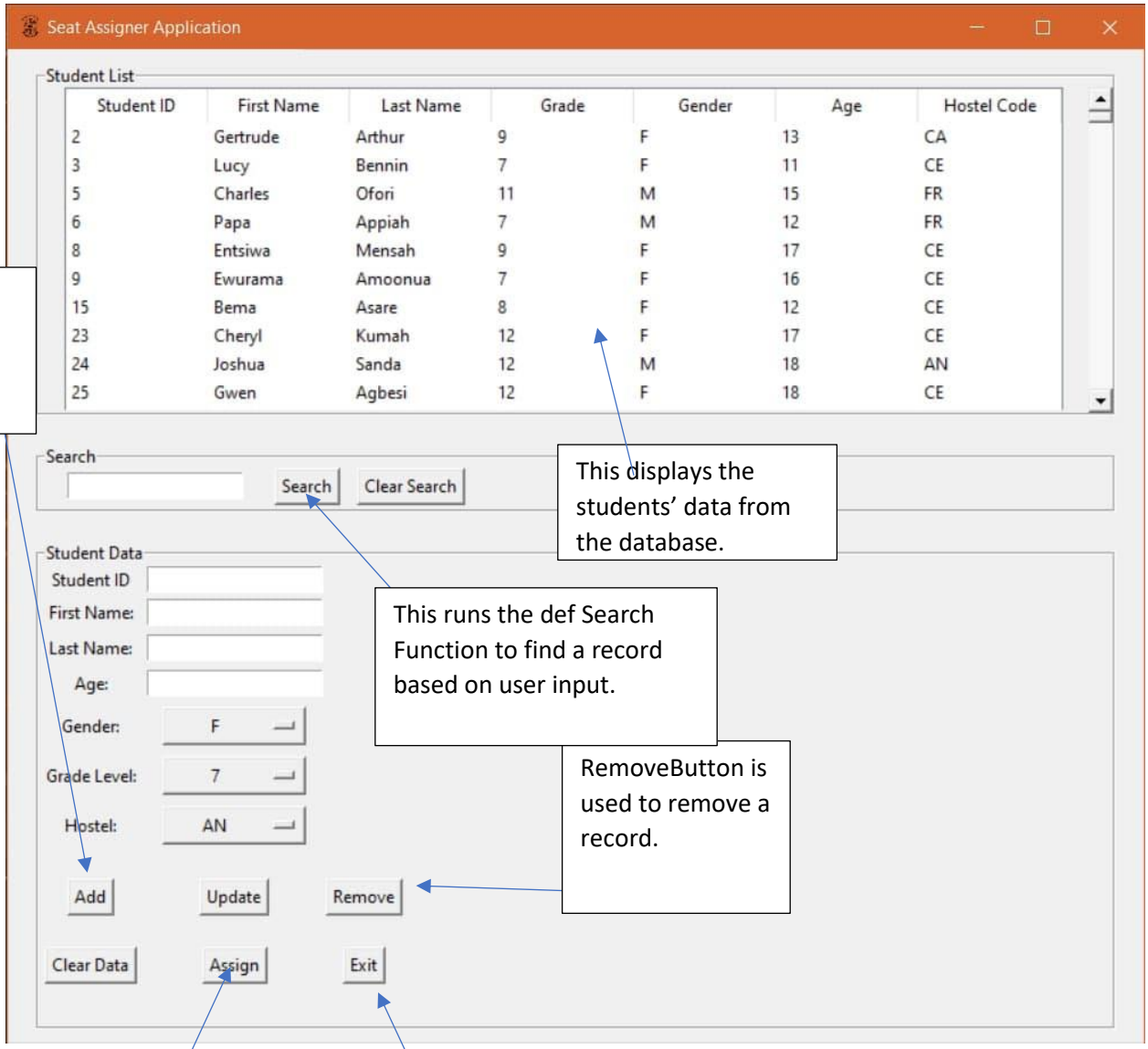
Criterion C:

Graphic User Interface.

The Graphical User Interface (GUI) was used to provide graphical elements like buttons and windows to allow the user to navigate through the program freely. This method of abstracting the code is so that the user does not have to worry about what certain lines of code can and cannot do.



AddButton used to add a record.

This displays the students' data from the database.

This runs the def Search Function to find a record based on user input.

RemoveButton is used to remove a record.

AssignButton is used to run the whole program of assigning students to tables.

Exitbutton is used to close the whole application.

GUI object manipulation:

The intuitive Tkinter library in python allows for the manipulation and placement of various elements such as buttons and radio buttons.

There are various code snippets throughout the program that controls various GUI element actions for the types of elements seen above. The first is the functionality of the “Exit” button, which closes the whole program.

tk.Button is for the Library to recognize the widget as a button and not anything else like a label.

```
#Adding an exit button
exit_button = tk.Button(label3, text='Exit', command=root.quit)
exit_button.grid(row=9, column=3, padx=2, pady=2)
```

The next one is the placement of the Add, Remove and Assign buttons on the program’s main window.

```
Add_button = tk.Button(label3, text="Add", command=addNew)
Add_button.grid(row=8, column=1, padx=2, pady=20)

#A button that updates a record
Update_Button = tk.Button(label3, text="Update", command=Update_St
Update_Button.grid(row=8, column=2, pady=20)

# Add an remove button that will run the delete function of the database
remove_button = tk.Button(label3, text="Remove", command=deleteStudent)
remove_button.grid(row=8, column=3, pady=20)

#This is the assign button, it does the assigns the tables
AssignButton = tk.Button(label3, text = "Assign", command = Assign)
AssignButton.grid(row=9, column=2)
```

Row, column, padx and pady allow for easy movement in a 2-dimensional array style.

Label3 ensures that the widget is in the correct frame. Each widget is attached to the main window, referref to as root.

Command creates the event to call the function in a different section of the program’s code.

Database Management:

I connected to the Access Database using the Pyodbc library. This allows me to make changes to the database from PyCharm. Database Management was used in order to facilitate the manipulation of database data from the application rather than having both the program and database open when using it.

r'Driver is used to read the database in order to make changes.

```
import pyodbc

conn = pyodbc.connect(
    r'Driver={Microsoft Access Driver (*.mdb, *.accdb)};DBQ=C:/Users/HP/source/repos/Criterion C/Testing Data.accdb;')
cursor = conn.cursor()
cursor.execute('select * from names_table')
```

Importing the pyodbc library so the IDE can compute the code that is written in it.

The cursor variable is used to make a cursor which will allow the program to go through the Database in order to read and write to and from it.

Event Handling:

The application has many event handlers which allow for its operation. Event handlers manage events, for example, when a button is clicked or when an input has been made. In this program, event handlers are mainly buttons that run functions when clicked. Event handlers in this application include ones which will help the user:

- Close the program.
- Add a record.
- Remove a record.
- Update a record.
- Assign the students to tables.

```
#Adding an exit button
exit_button = tk.Button(label3, text='Exit', command=root.quit)
exit_button.grid(row=9, column=3, padx=2, pady=2)
```

The “command=root.quit” handles the event of the user clicking on the “Exit button”. the command root.quit will cause the whole program to close.

```
#Add adds a new record to the database
Add_button = tk.Button(label3, text="Add", command=addNew)
Add_button.grid(row=8, column=1, padx=2, pady=20)
```

command=addNew here is the event handler that calls on the addNew function. It reads the database and writes the new record using the inputs made by the user.

```
#A button that updates a record
Update_Button = tk.Button(label3, text="Update", command=Update_Student)
Update_Button.grid(row=8, column=2, pady=20)

# Add an remove button that will run the delete function of the database
remove_button = tk.Button(label3, text="Remove", command=deleteStudent)
remove_button.grid(row=8, column=3, pady=20)
```

The commands run the respective function. The event here is clicking the button and is handled by making a change in the display table’s view.

Functions:

I included certain functions to ensure that I did not have to write long lines of code for a simple command. I also included the functions to clean up the code I had written, to make it more presentable to view. Some of the functions include an Add and a Remove function which when called using the event handlers from above, will allow the user to interact with the GUI and make changes in the database and application. There are also options to use an Update function and clear a search using function in this program.

```
def addStudent(FN, LN, Age, Grade, Gender, HC):
    cursor = conn.cursor()
    cursor.execute('SELECT * from Students')
    sql = "INSERT INTO Students (FirstName, LastName, Grade, Gender, Age, HostelCode) "
    sql = sql + "VALUES('" + FN + "','" + LN + "','" + str(Grade) + "','" + str(Gender) + "','" + str(
        Age) + "','" + str(HC) + "')"
    cursor.execute(sql)
    Clear()
    conn.commit()

def addNew():
    try:
        int(age_entry.get())
    except ValueError:
        warning = messagebox.showerror("Error", "Age Should be a Number.")
        if warning == 0:
            root.destroy()
        else:
            return True
    else:
        confirmation = messagebox.askyesno("Are you sure?", "Are you sure you want to add this student's record?")
        if confirmation == 1:
            FName = First_Name_entry.get()
            LName = Last_Name_entry.get()
            Age = age_entry.get()
            Grade = grade.get()
            Gender = gender.get()
            HCode = hostel.get()
            addStudent(FName, LName, Age, Grade, Gender, HCode)
            success_lbl.config(text="Student Successfully Added!", fg='black')
        else:
            return True
```

There are certain parameters needed for the function to run. These are satisfied in the SQL statement section of the function.

The function allows all the code to be called once instead of being written multiple time to save processing time and disk space.

This Add function allows the program to call it from the event handler using command=addNew in order to add a new record. The functionality applies to the Remove, Update and Clear functions.

```python
#PARAMETERS ARE ID, FirstName, LastName, Grade, Gender, Age, HostelCode
def deleteStudent():
    studentID = identityEntry.get()
    if messagebox.askyesno("Confirm Delete?", "Are you sure you want to delete this record?"):
        #delete = "DELETE FROM Students WHERE ID = " + studentID
        delete = "DELETE FROM Students WHERE ID LIKE '%" + str(studentID) + "%'"
        cursor.execute(delete)
        conn.commit()
        Clear()
    else:
        return True
```

```python
#Updating Students Details
def Update_Student():
    studentID = identityEntry.get()
    FirstName = First_Name_entry.get()
    LastName = Last_Name_entry.get()
    Grade = int(grade.get())
    Gender = str(gender.get())
    Age = int(age_entry.get())
    Hostel = str(hostel.get())
    print(studentID, FirstName, LastName, Grade, Gender, Age, Hostel)

    if messagebox.askyesno("Confirm Please", "Are you sure you want to update this student's record?"):
        query = "UPDATE Students SET FirstName = '" + FirstName + "'," + " LastName = '" + LastName + "'," + \
                " Grade = " + str(Grade) + "," + " Gender = '" + Gender + "'," + " Age = " + str(Age) + "," + \
                " HostelCode = '" + Hostel + "' WHERE ID = " + studentID
        print(query)
        cursor.execute(query)
        conn.commit()
        Clear()
    else:
        return True
```

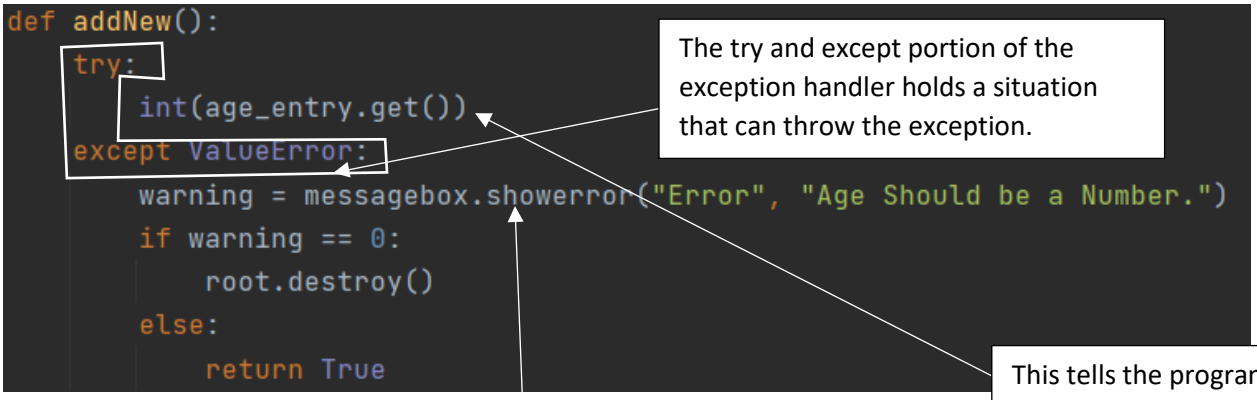
```
#Updating the treeview table
def Update(rows):
    tree.delete(*tree.get_children())
    for row in rows:
        display = []
        for item in row:
            display.append(str(item))
        tree.insert("", "end", values= display)

#Treeview Clearing
def Clear():
    searchEntry.delete(0, 'end')
    query = "SELECT * from Students"
    cursor.execute(query)
    rows = cursor.fetchall()
    Update(rows)
```

A sub function which opens a new window to ask the user if they are sure that they want to delete a record from the database. The advantage of using functions in the creation of this application is that it saves time by only writing the code once so it can be called at any time that it is needed.

Exception Handling:

These were added to stop the program from running if there is any invalid data in it. It responds to unrecognizable data types which were input by the user. The exception handler is used as a method of handling an error created by inputting wrong data using the try…except block to do so.



The try and except portion of the exception handler holds a situation that can throw the exception.

This tells the program that the input must be an integer.

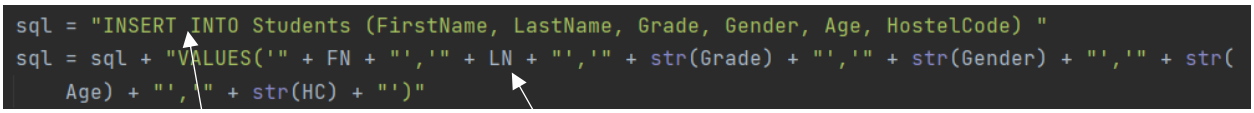
The error is handled with an error message which prompts the user to make a correction to the data.

Standard Query Language:

This was included as a way to read data from the database and return values to PyCharm. SQL is used to manage databases by allowing certain processes like Selection, Insertion, Updates, Deletions and Joins.

Select and Insert Query:

The select statement will retrieve a row (a record) from within a database table that is specified. The SELECT statement is used multiple time in this program. It is used for selecting students to assign them to tables and to add and remove them from the database.



This query inputs data provided by the user into the Students table in the database.

The Values are the parameters from the function under the functions section. The values are gotten from user inputs that are stored in variables. (The parameters are variables.)

Delete Query:

The delete query will delete a row (a record) from a specified table. The following query was used to remove students from the database.

```
delete = "DELETE FROM Students WHERE ID LIKE '%" + str(studentID) + "%'"
```

The LIKE looks for similarities in the user input and the data stored in the database.

Join Query:

The join query was used to join two tables in a one-to-one relationship. The type of join depends on what kind of data is being joined. For this program and Inner Join was used because the data (ID) is the same in both the Students and TableGroups tables.

```
sqlStmt = "SELECT S.FirstName, S.LastName, S.Grade FROM Students S INNER JOIN TableGroups T ON S.ID = T.ID "
sqlStmt = sqlStmt + " WHERE T.GroupNumber = " + str(GNumber)
```

The inner join statement mimics the intersection of 2 sets of data so only the data that match each other will be operated on.

Update Query:

```
query = "UPDATE Students SET FirstName = '" + FirstName + "'," + " LastName = '" + LastName + "'," + \
        " Grade = " + str(Grade) + "," + " Gender = '" + Gender + "'," + " Age = " + str(Age) + "," + \
        " HostelCode = '" + Hostel + "' WHERE ID = " + studentID
```

The Update statement sets the already stored as a new piece of data without changing its location or value (unless it is an integer). Updates were used to provide a failsafe for the user in case they entered the wrong data for a certain student.

Search Query

```
query = "SELECT * FROM Students WHERE FirstName LIKE '%" + user + "%' OR LastName LIKE '%" + user + "%' OR Grade LIKE '%" + user + "%'" \
    "OR Gender LIKE '%" + user + "%' OR HostelCode LIKE '%" + user + "%'"
```

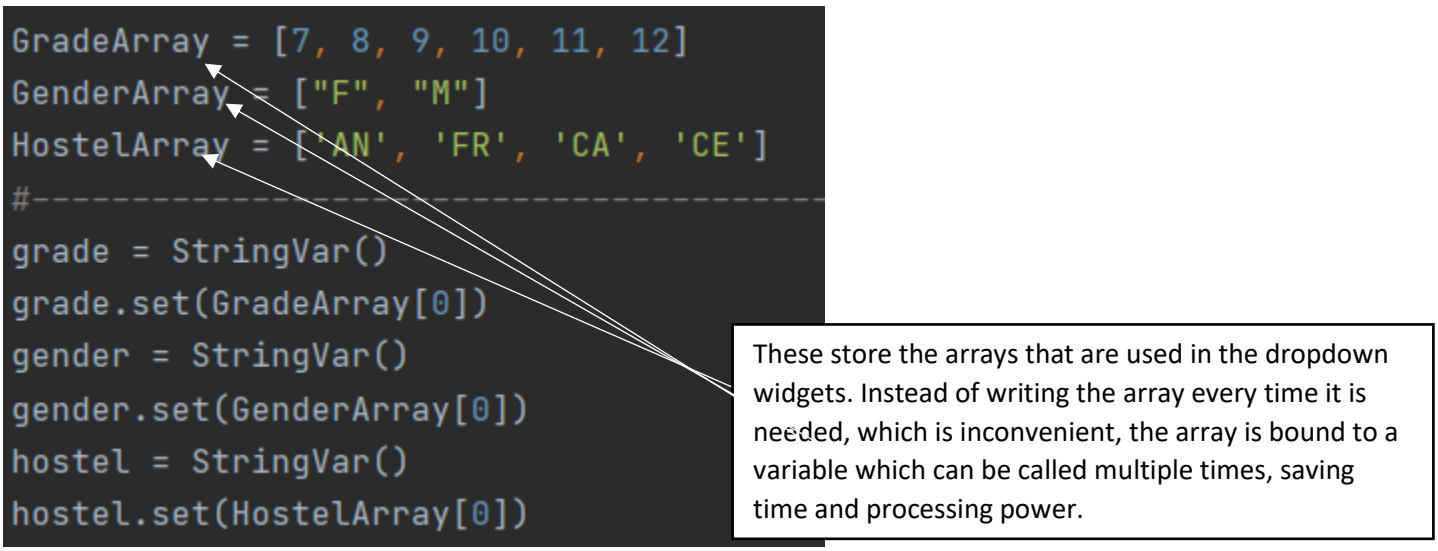
Search Queries were used to find certain data that is similar to that the user inputs. This is used in case the user forgets a student ID and only remembers their name or grade.

Variables:

These are storage locations paired with an associated symbolic name which contains some known or unknown quantity of information referred to as a value. Variables were used in this program to store long SQL queries to make the code more presentable and easier to understand.

```
sql = "INSERT INTO Students (FirstName, LastName, Grade, Gender, Age, HostelCode) "
sql = sql + "VALUES('" + FN + "','" + LN + "','" + str(Grade) + "','" + Gender + "','" + str(
    Age) + "','" + HC + "')"
cursor.execute(sql)
```

Variables were also used to store string variables which will allow for concatenation when writing SQL statements. For example, the dropdown widgets use variables to store their contents in grade, gender, and hostel. Making them string variables makes operation on those values easier



Loops:

These cause iterations of a specific event until it is changed. It can either be fixed in the case of a for loop or it can go on until a certain parameter is met or changed in the case of a while loop. The program uses for loops and nested loops in order to go through and group students in order to place them on tables. It also uses loops to iterate between records and write their names to files.

```
maxTables = 0
cursor.execute("select count(*) from Students")
results = cursor.fetchall()
for row in results:
    maximum = row[0] / 7
    maxTables = int(math.ceil(maximum))
```

The for loop would keep the program iterating until all the rows have been searched.

```
# First loop in order to assign group numbers to all existing records on the database
genderIndex = ["F", "M"]
tableNumber = 1
for grade in range(7, 13):
    for gender in genderIndex:
        sql = "SELECT ID from Students WHERE Grade = " + str(grade) + " AND Gender = '" + gender + "'"
        cursor.execute(sql)
        results = cursor.fetchall()
        for row in results:
            tableNumber = tableNumber + 1
            if tableNumber > maxTables:
                tableNumber = 1
            cursor.execute(
                "INSERT INTO TableGroups (ID, GroupNumber, Grade) VALUES(" + str(row[0]) + "," + str(tableNumber) + "," + str(grade) + ")")
conn.commit()
```

These nested loops will make the program run the inner loop first before running the outer loop. This makes the program group students according to gender and grade levels to make it as equal as possible.

```
file_write = open('groups.txt', 'w')
for GNumber in range(1, maxTables + 1):
    file_write.write('\n')
    file_write.write("TABLE NUMBER" + " " + str(GNumber) + " " + '\n')
    sqlStmt = "SELECT S.FirstName, S.LastName, S.Grade FROM Students S INNER JOIN TableGroups T ON S.ID = T.ID "
    sqlStmt = sqlStmt + " WHERE T.GroupNumber = " + str(GNumber)
    cursor.execute(sqlStmt)
    results = cursor.fetchall()

    count = 1
    for row in results:
        file_write.write(" " + str(count) + ". ")
        file_write.write(row[0] + " " + row[1] + " " + "(Grade " + str(row[2]) + ")" + '\n')
        done_lbl.config(text="Tables Assigned Successfully!", fg="green")
        count = count+1
file_write.close()
```

The loops help iterate table numbers until it reaches the last table that was calculated. The inner loop then iterates between the students in the database and writes their names and grades in a file.

Data Structures:

Data Tables:

This is any display in any tabular form with rows and columns that have names. This is like the physical schema of the database. Each row is an individual record, and the columns are the fields names. If we run an output query it will select the data we want and put it in a data table.

Select FirstName, LastName, Grade

From Students

Where Age < 11

We get a Data table like this:

FirstName	**LastName**	**Grade**
Maame	Johnson	8

Arrays: This is a way of storing values or strings in adjoining blocks of data in a contiguous manner. This is a set number of objects that when called will only have the values available in it for operation in the program. There should be about 3 arrays in the program.

GenderArray= [“F”, “M”]

GradeArray = [7, 8, 9, 10, 11, 12]

HostelArray = [“AN”, “FR”, “CE”, “CA”]

Bibliography

Doron (2018) *How to Connect Python to MS Access Database using pyodbc, Datatofish.com*. Available at: https://datatofish.com/how-to-connect-python-to-ms-access-database-using-pyodbc/ (Accessed: April 7, 2021).

Kumar, B. (2020) *Python array with examples - Python guides, Pythonguides.com*. Available at: https://pythonguides.com/python-array/ (Accessed: April 7, 2021).

List of Tkinter Widgets - with examples - CodersLegacy (2020) *Coderslegacy.com*. Available at: https://coderslegacy.com/python/list-of-tkinter-widgets/ (Accessed: April 7, 2021).

Python, R. (2020) *Python GUI Programming With Tkinter*, *Realpython.com*. Real Python. Available at: https://realpython.com/python-gui-tkinter/ (Accessed: April 7, 2021).

SQL SELECT COUNT (no date) *Sql-statements.com*. Available at: http://www.sql-statements.com/sql-select-count.html (Accessed: April 7, 2021).

(No date) *Datacamp.com*. Available at: https://www.datacamp.com/community/tutorials/loops-python-tutorial (Accessed: April 7, 2021).