# COMP1811 – Python Project Report

| **Name** | **Morales Huaygua, Luis David** | **Student ID** | **001137079** |
| --- | --- | --- | --- |
| **Partner’s name** | **Muthaliph, Gnei Tiara Rahiza** | **Partner SIDs** | **001166581** |

# Brief statement of features you have completed

*(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)*

***Indicate the feature each partner implemented by replacing “developed by” in red below with partner name****.*

| 1.1 Circle the parts of the coursework you have **fully completed and are fully working**. Please be accurate. | **Features *[Luis David] [Gnei Tiara]*** | **F1:** i ii  iii  **F2:** i ii  iii |
| --- | --- | --- |
| 1.2 Circle the parts of the coursework you have **partly completed or are partly working.** | **Features** | **F1:** i ii  iii  **F2:** i ii  iii |
| Briefly explain your answer if you circled any parts in 1.2 | | |

# Concise List of Bugs and Weaknesses

*A concise list of bugs and/or weaknesses in your work (if you don't think there are any, then say so). Bugs that are declared in this list will lose you fewer marks than ones that you don't declare! (****100-200 word****, but word count depends heavily on the number of bugs and weaknesses identified.)*

*(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)*

## Bugs

*List each bug plus a brief description*

I believe I have no bugs in the code and the app

## Weaknesses

*List each weakness plus a brief description*

I believe the app has no weaknesses. A user can register, take the quiz (when develop) and exit the quiz. The user can also enter the admin page to add, delete, and update modules and questions. One possible weakness could be the app dealing with more than 30+ modules as this can overflow the tkinter app. However, this could be also solved by implementing a search bar where the user can look up modules by their names.

Another weakness is that the user cannot save its profile when taking the quiz. I was not directly working with this feature as it was not my part but I believe this problem could be solve by just storing the username as an attribute in a class. It could also be requested for the user to type it once each quiz has been taken. This username, along with its score would be stored to a userResults table in the DB.

# Description of the features implemented

*Describe your implementation of the required features and how well do they work. Provide some exposition of the design decisions made and indicate how the features developed were integrated.   
(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)*

**Add module:**

This feature consists of one button that when clicked it raise a new tkinter frame window. In this new window you first need to select a starting question type. This starting question is asked mainly because it is needed to go to the edit module frame.

Once the type of the question is selected, another window frame will pop up with entries for the question name, score, answers, and possible answers (except for the T/F). When all entries have been filled, the user can click on submit and all the user input will be pushed to their respective tables in the database. This was done by defining multiple functions that act as “queries calls” for each operation (add, delete, update, select).

Finally, the feedback window frame will pop up an entry for each answer. Again here, functions queries were very helpful to fetch the answers and render it on the frame.

**Delete module:**

This feature consists of one button that when clicked it raises a new tkinter frame window. This window contains a drop-down list showing all the current modules from the database along with a button at the right. To delete a module, the user simply would need to choose one module and click on “Erase”.

It is worth to mention that this button won’t work unless there at least 1 module in the database.

**Edit/Delete question and module name change:**

Once a module has been added to the database, the name of the module will appear as a button on the body of the AdminHomePage class/Frame. This button when clicked will pop up a new window where the user can edit the question name, score and answers. The module name can also be edited.

Questions can also be deleted and added from this “Edit Module window”. There will be another drop down list and a “edit” button where the user can one question from the list of questions that the module has. When clicked in edit, a window frame will pop up. This new window frame will contain all the information in **entries** from the question in separate sections with their respective “Apply” button.

To add a new question to a module, the user would need to click on the respective module and click on the Add question button on the “edit module window”. The rest will be same as in the “Add module section” except the module name that will already be selected and will be unchangeable.

**Application Interactive GUI:**

This part consists of an extra Login Class/Frame where the user can decide to go to the admin or user features depending on the username and password.

# Classes and OOP Features

*List all the classes used in your program and include the attributes and behaviours for each. You may use a class diagram to illustrate these classes. Your narrative for section 3.2 should describe the design decisions you made and the OOP techniques used. Each partner must list the classes they developed separately and provide an exposition on the choice of classes, class design and OOP features implemented. (****200-400 words for each partner****). (THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)*

## Diagram Description automatically generatedClasses Used

Figure 1 Class Design

## Brief Explanation of Class Design and OOP Features Used

I used three classes for this project and multiple tkinter windows. I decided to use tkinter windows for almost every feature (add module, delete module…) because it made it much simpler to pass information from one window frame to another window frame than the conventional class attributes. This is the reason I do not use attributes in each class. However, I believe the behavior part of each class was good, as I did use a lot of them.

One OPP technique that particularly that stand out in the app is the inheritance technique. Both classes, (adminHomeFeatures and Log In Features) have availability to use part of the behavior of QuizzAPP class such as to change to another frame if necessary.

In terms of the tkinter windows, I did not find any solution to pass information from class to class without having to access the main (QuizzApp) class. This means from the adminHomeFeatures, we can only access attributes from the QuizAPP. Therefore, I used tkinter windows. Also, I could not figure out how to build a sort of tree structure with the classes. For instance, I wanted to build a child class for AdminHomeFeatures lass, but I couldn’t. I would need to build a child class for QuizzApp instead, so the only solution I came out with was using tkinter windows inside the AdminHomeFeatures. I later discovered how class would inherit their attributes while still being frames. Unfortunately, I could not re do all the work I’ve done so far and deliver on time.

# Code for the Classes Created

*Add the* ***code for each of the classes you have implemented yourself*** *here. If you have contributed to parts of classes, please highlight those parts in a different colour. Copy and paste relevant code - actual code please, no screenshots! Make it easy for the tutor to read. Add explanation if necessary – though your in-code comments should be clear enough. You will lose marks if screenshots are provided instead of code.*

*(COMPLETE THIS SECTION INDIVIDUALLY – only list the code for the classes you developed individually. DO NOT provide a listing of the entire code. You will be marked down if a full code listing is provided.)*

## Class …

1. class AdminHomePage(tk.Frame):  
    def \_\_init\_\_(self, parent, controller):  
    tk.Frame.\_\_init\_\_(self, parent)  
    self.isActive = False  
     
     
    head = tk.LabelFrame(self, text="Admin Page", fg="white", bg='#5D9DE5', font=('Helvetica', 28, 'bold'), bd=1)  
    head.pack(fill='both', expand='yes', padx=20, pady=10)  
     
    titleLabel = tk.Label(head, text="Current Modules", fg="white", bg='#5D9DE5', font=('Helvetica', 20, 'bold'))  
    titleLabel.grid(row=0, column=0, pady=20)  
    def closeSystem():  
    self.destroy()  
    exit\_l = tk.Button(head, text="Exit", fg="white", bg='#5D9DE5', font=('Helvetica', 20, 'bold'), command=lambda:closeSystem())  
    exit\_l.place(x=550, y=540)  
     
    def update\_option\_menu(m\_quest\_to\_del, curr\_mod\_name, quest\_to\_del):  
    m = m\_quest\_to\_del['menu']  
    m.delete(0, 'end')  
    list\_of\_quest = fetch\_all\_quest(curr\_mod\_name)  
    for string in list\_of\_quest:  
    m.add\_command(label=string, command=lambda value=string: quest\_to\_del.set(value))  
     
    def onlyDeleteBtnModules():  
    row = 2  
    col = 0  
    count = len(fetchModules())  
    print(count, 'sss')  
    while row < 7 and col < 2 and count > 0:  
    a = head.grid\_slaves(row, col)  
    if len(a) > 0:  
    a[0].destroy()  
    count -= 1  
    row += 1  
    if row == 7:  
    row = 2  
    col += 1  
    self.isActive = False  
    return  
     
    # EDIT QUESTIONS MODE NAME  
    def edit\_quest\_frame(mod\_name, quest\_name):  
     
    window = tk.Tk()  
    window.resizable(0, 0)  
    window.geometry("700x600")  
    m\_quest\_features = tk.LabelFrame(window, text="Admin Page - Edit Question",fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'),  
    bd=1)  
    m\_quest\_features.pack(fill='both', expand='yes', padx=20, pady=10)  
     
    # Find type of quest and id:  
    type\_of\_q = find\_quest\_type(quest\_name)  
    quest\_id = find\_quest\_id(quest\_name)  
     
    quest\_module = tk.Label(m\_quest\_features, text="Module: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    quest\_module.place(x=10, y=10)  
     
    quest\_module\_title = tk.Label(m\_quest\_features, text=mod\_name, fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    quest\_module\_title.place(x=180, y=10)  
     
    quest\_title = tk.Label(m\_quest\_features, text="Question Name: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    quest\_title.place(x=10, y=60)  
     
    e\_quest\_title = tk.Entry(m\_quest\_features, width=30, bd=5)  
    e\_quest\_title.place(x=150, y=60)  
    e\_quest\_title.insert(tk.END, quest\_name)  
     
    quest\_t\_btn = tk.Button(m\_quest\_features,fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), text="change", command=lambda: update\_quest(  
    quest\_id,  
    'quest\_name',  
    e\_quest\_title.get(),  
    m\_quest\_features  
    ))  
    quest\_t\_btn.place(x=360, y=57)  
     
    if type\_of\_q == 'tf':  
    l\_ans = tk.Label(m\_quest\_features, text="Answer: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    l\_ans.place(x=10, y=100)  
    get\_ans\_tf = find\_ans\_from\_quest(quest\_id)  
    e\_ans\_tf = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_ans\_tf.place(x=10, y=130)  
    e\_ans\_tf.insert(tk.END, get\_ans\_tf)  
    # Update answer column  
    ans\_tf\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
    command=lambda: update\_quest(quest\_id, 'answer', e\_ans\_tf.get(), m\_quest\_features))  
    ans\_tf\_btn.place(x=10, y=160)  
     
    get\_inc\_ans\_tf = find\_inc\_ans\_from\_quest(quest\_id)  
    l\_inc\_ans = tk.Label(m\_quest\_features, text="Incorrect Answer: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    l\_inc\_ans.place(x=10, y=200)  
     
    e\_inc\_ans = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_inc\_ans.place(x=10, y=230)  
    e\_inc\_ans.insert(tk.END, get\_inc\_ans\_tf)  
    # Update possible answer column  
     
    ans\_tf\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
    command=lambda: update\_quest(quest\_id, 'possible\_answer', e\_inc\_ans.get(), m\_quest\_features))  
    ans\_tf\_btn.place(x=10, y=260)  
     
    # Update question Mark  
    get\_quest\_mark = find\_quest\_mark(quest\_id)  
    l\_quest\_mark = tk.Label(m\_quest\_features, text="Question Mark: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    l\_quest\_mark.place(x=10, y=290)  
    e\_quest\_mark = tk.Entry(m\_quest\_features, width=8, bd=5)  
    e\_quest\_mark.place(x=10, y=320)  
    e\_quest\_mark.insert(tk.END, get\_quest\_mark)  
     
    q\_m\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
    command=lambda: update\_quest(quest\_id, 'quest\_mark', e\_quest\_mark.get(), m\_quest\_features))  
    q\_m\_btn.place(x=10, y=350)  
    if type\_of\_q == 'mcq':  
    l\_ans = tk.Label(m\_quest\_features, text="Answer: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    l\_ans.place(x=10, y=100)  
    get\_ans = find\_ans\_from\_quest(quest\_id)  
    len\_of\_ans = len(get\_ans)  
    if len\_of\_ans == 1:  
    e\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_ans\_1.place(x=10, y=130)  
    e\_ans\_1.insert(tk.END, get\_ans[0])  
    elif len\_of\_ans == 2:  
    e\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_ans\_1.place(x=10, y=130)  
    e\_ans\_1.insert(tk.END, get\_ans[0])  
     
    e\_ans\_2 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_ans\_2.place(x=10, y=160)  
    e\_ans\_2.insert(tk.END, get\_ans[1])  
    elif len\_of\_ans == 3:  
    e\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_ans\_1.place(x=10, y=130)  
    e\_ans\_1.insert(tk.END, get\_ans[0])  
     
    e\_ans\_2 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_ans\_2.place(x=10, y=160)  
    e\_ans\_2.insert(tk.END, get\_ans[1])  
     
    e\_ans\_3 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_ans\_3.place(x=10, y=190)  
    e\_ans\_3.insert(tk.END, get\_ans[2])  
     
    get\_inc\_ans = find\_inc\_ans\_from\_quest(quest\_id)  
    l\_inc\_ans = tk.Label(m\_quest\_features, text="Incorrect Answers: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    l\_inc\_ans.place(x=10, y=250)  
    len\_of\_inc\_ans = len(get\_inc\_ans)  
     
    if len\_of\_inc\_ans == 1:  
    e\_inc\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_inc\_ans\_1.place(x=10, y=280)  
    e\_inc\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
    if len\_of\_inc\_ans == 2:  
    e\_inc\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_inc\_ans\_1.place(x=10, y=280)  
    e\_inc\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
     
    e\_inc\_ans\_2 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_inc\_ans\_2.place(x=10, y=310)  
    e\_inc\_ans\_2.insert(tk.END, get\_inc\_ans[1])  
     
    if len\_of\_inc\_ans == 3:  
    e\_inc\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_inc\_ans\_1.place(x=10, y=280)  
    e\_inc\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
     
    e\_inc\_ans\_2 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_inc\_ans\_2.place(x=10, y=310)  
    e\_inc\_ans\_2.insert(tk.END, get\_inc\_ans[1])  
     
    e\_inc\_ans\_3 = tk.Entry(m\_quest\_features, width=20, bd=5)  
    e\_inc\_ans\_3.place(x=10, y=340)  
    e\_inc\_ans\_3.insert(tk.END, get\_inc\_ans[2])  
     
    def gather\_inc\_ans():  
    if len\_of\_inc\_ans == 1:  
    return ",".join([e\_inc\_ans\_1.get()])  
    if len\_of\_inc\_ans == 2:  
     
    return ",".join([e\_inc\_ans\_1.get(), e\_inc\_ans\_2.get()])  
    if len\_of\_inc\_ans == 3:  
    return ",".join([e\_inc\_ans\_1.get(), e\_inc\_ans\_2.get(), e\_inc\_ans\_3.get()])  
     
    def gather\_ans():  
    if len\_of\_ans == 1:  
    return ",".join([e\_ans\_1.get()])  
    if len\_of\_ans == 2:  
    return ",".join([e\_ans\_1.get(), e\_ans\_2.get()])  
    if len\_of\_ans == 3:  
    return ",".join([e\_ans\_1.get(), e\_ans\_2.get(), e\_ans\_3.get()])  
     
    ans\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
    command=lambda: update\_quest(quest\_id, 'answer', gather\_ans(), m\_quest\_features))  
    ans\_btn.place(x=10, y=220)  
     
    inc\_ans\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
    command=lambda: update\_quest(  
    quest\_id,  
    'possible\_answers',  
    gather\_inc\_ans(), m\_quest\_features))  
    inc\_ans\_btn.place(x=10, y=370)  
     
    get\_quest\_mark = find\_quest\_mark(quest\_id)  
    l\_quest\_mark = tk.Label(m\_quest\_features, text="Question Mark: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    l\_quest\_mark.place(x=10, y=410)  
    e\_quest\_mark = tk.Entry(m\_quest\_features, width=8, bd=5)  
    e\_quest\_mark.place(x=150, y=410)  
    e\_quest\_mark.insert(tk.END, get\_quest\_mark)  
     
    q\_m\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
    command=lambda: update\_quest(quest\_id, 'quest\_mark', e\_quest\_mark.get(), m\_quest\_features))  
    q\_m\_btn.place(x=220, y=410)  
     
    if type\_of\_q == 'bm':  
    possible\_ans = tk.Label(m\_quest\_features, text="Possible Answers: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    possible\_ans.place(x=10, y=100)  
    get\_inc\_ans = find\_inc\_ans\_from\_quest(quest\_id)  
    get\_ans = ",".join(find\_ans\_from\_quest(quest\_id)).split(',')  
    l\_inc = len(get\_inc\_ans)  
     
    if l\_inc == 1:  
    e\_p\_ans\_1 = tk.Entry(m\_quest\_features, width=15, bd=5)  
    e\_p\_ans\_1.place(x=10, y=130)  
    e\_p\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
     
    bma\_e\_p\_ans1 = tk.StringVar(m\_quest\_features)  
    bma\_e\_p\_ans1.set(get\_father\_from\_child(get\_inc\_ans[0])) # default value  
     
    curr\_bma\_e\_p\_ans1 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans1, \*get\_ans)  
    curr\_bma\_e\_p\_ans1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_1 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans1.menuname)  
    curr\_1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_bma\_e\_p\_ans1.place(x=150, y=130)  
     
    if l\_inc == 2:  
    e\_p\_ans\_1 = tk.Entry(m\_quest\_features, width=15, bd=5)  
    e\_p\_ans\_1.place(x=10, y=130)  
    e\_p\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
     
    e\_p\_ans\_2 = tk.Entry(m\_quest\_features, width=15, bd=5)  
    e\_p\_ans\_2.place(x=10, y=180)  
    e\_p\_ans\_2.insert(tk.END, get\_inc\_ans[1])  
     
    bma\_e\_p\_ans1 = tk.StringVar(m\_quest\_features)  
    bma\_e\_p\_ans1.set(get\_father\_from\_child(get\_inc\_ans[0])) # default value  
     
    curr\_bma\_e\_p\_ans1 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans1, \*get\_ans)  
    curr\_bma\_e\_p\_ans1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_1 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans1.menuname)  
    curr\_1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_bma\_e\_p\_ans1.place(x=150, y=130)  
     
    bma\_e\_p\_ans2 = tk.StringVar(m\_quest\_features)  
    bma\_e\_p\_ans2.set(get\_father\_from\_child(get\_inc\_ans[1])) # default value  
     
    curr\_bma\_e\_p\_ans2 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans2, \*get\_ans)  
    curr\_bma\_e\_p\_ans2.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_2 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans2.menuname)  
    curr\_2.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_bma\_e\_p\_ans2.place(x=150, y=180)  
     
    if l\_inc == 3:  
     
    e\_p\_ans\_1 = tk.Entry(m\_quest\_features, width=15, bd=5)  
    e\_p\_ans\_1.place(x=10, y=130)  
    e\_p\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
     
    e\_p\_ans\_2 = tk.Entry(m\_quest\_features, width=15, bd=5)  
    e\_p\_ans\_2.place(x=10, y=180)  
    e\_p\_ans\_2.insert(tk.END, get\_inc\_ans[1])  
     
    e\_p\_ans\_3 = tk.Entry(m\_quest\_features, width=15, bd=5)  
    e\_p\_ans\_3.place(x=10, y=230)  
    e\_p\_ans\_3.insert(tk.END, get\_inc\_ans[2])  
     
    bma\_e\_p\_ans1 = tk.StringVar(m\_quest\_features)  
    bma\_e\_p\_ans1.set(get\_father\_from\_child(get\_inc\_ans[0])) # default value  
     
    curr\_bma\_e\_p\_ans1 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans1, \*get\_ans)  
    curr\_bma\_e\_p\_ans1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_1 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans1.menuname)  
    curr\_1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_bma\_e\_p\_ans1.place(x=150, y=130)  
     
    bma\_e\_p\_ans2 = tk.StringVar(m\_quest\_features)  
    bma\_e\_p\_ans2.set(get\_father\_from\_child(get\_inc\_ans[1])) # default value  
     
    curr\_bma\_e\_p\_ans2 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans2, \*get\_ans)  
    curr\_bma\_e\_p\_ans2.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_2 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans2.menuname)  
    curr\_2.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_bma\_e\_p\_ans2.place(x=150, y=180)  
     
    bma\_e\_p\_ans3 = tk.StringVar(m\_quest\_features)  
    bma\_e\_p\_ans3.set(get\_father\_from\_child(get\_inc\_ans[2])) # default value  
     
    curr\_bma\_e\_p\_ans3 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans3, \*get\_ans)  
    curr\_bma\_e\_p\_ans3.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_3 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans3.menuname)  
    curr\_3.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_bma\_e\_p\_ans3.place(x=150, y=230)  
     
    def update\_bma\_ans():  
    p\_ans = []  
    ans = []  
    m\_id = findModId(mod\_name)  
     
    if l\_inc == 1:  
    p\_ans.append(e\_p\_ans\_1.get())  
    ans.append(bma\_e\_p\_ans1.get())  
    if l\_inc == 2:  
    p\_ans.append(e\_p\_ans\_1.get())  
    p\_ans.append(e\_p\_ans\_2.get())  
    ans.append(bma\_e\_p\_ans1.get())  
    ans.append(bma\_e\_p\_ans2.get())  
    if l\_inc == 3:  
    p\_ans.append(e\_p\_ans\_1.get())  
    p\_ans.append(e\_p\_ans\_2.get())  
    p\_ans.append(e\_p\_ans\_3.get())  
    ans.append(bma\_e\_p\_ans1.get())  
    ans.append(bma\_e\_p\_ans2.get())  
    ans.append(bma\_e\_p\_ans3.get())  
     
    i = 0  
     
    del\_bma\_rows(quest\_id)  
     
    while i < len(p\_ans):  
    add\_ans\_to\_bma(quest\_id, ans[i], p\_ans[i], m\_id)  
    i += 1  
    # return  
    del\_feed\_quest\_from\_db(quest\_id)  
    add\_feed\_frame(quest\_id, p\_ans, False, m\_id, True)  
    update\_quest(quest\_id, 'possible\_answers', ",".join(p\_ans), m\_quest\_features)  
     
    btn\_sub = tk.Button(m\_quest\_features, text="Submit", command=lambda:  
    update\_bma\_ans())  
    btn\_sub.place(x=10, y=260)  
     
    # Change mod name frame  
    def change\_mod\_name(curr\_mod\_name):  
    window = tk.Tk()  
    window.resizable(0, 0)  
    window.geometry("700x600")  
    mod\_features = tk.LabelFrame(window, text="Admin Page - Edit Module", fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'), bd=1)  
    mod\_features.pack(fill='both', expand='yes', padx=20, pady=10)  
     
    t\_new\_name = tk.Label(mod\_features, text="New Module Name: ", font=('Helvetica', 13, 'bold'), fg="white", bg='#5D9DE5')  
    t\_new\_name.place(x=10, y=10)  
    new\_name\_entry = tk.Entry(mod\_features, font=('Helvetica', 13, 'bold'), borderwidth=5, width=22, bd=5)  
    new\_name\_entry.place(x=10, y=50)  
    refresh\_btn = tk.Button(mod\_features, text="Refresh",fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), width=10,  
    command=lambda:  
    update\_option\_menu(curr\_quest\_to\_del\_m,  
    curr\_mod\_name,  
    curr\_quest\_to\_del  
    ))  
    refresh\_btn.place(x=550, y=10)  
     
    def execute\_mod\_change(curr\_name, desired\_name, frame=False):  
    if desired\_name == '':  
    messagebox.showinfo("Error",  
    "The field must have been left blank. Please fill the field you edited.",  
    parent=frame)  
    return  
    update\_mod\_name\_in\_db(curr\_name, desired\_name)  
    delBtnModsAndUpdate()  
    window.destroy()  
     
    new\_name\_sub = tk.Button(mod\_features, text="submit", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), width=7,  
    command=lambda: execute\_mod\_change(curr\_mod\_name, new\_name\_entry.get(), mod\_features))  
    new\_name\_sub.place(x=240, y=50)  
     
    # Add Question:  
     
    mod\_add\_quest\_l = tk.Label(mod\_features, text="Add new question to this module: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    mod\_add\_quest\_l.place(x=10, y=125)  
     
    add\_quest\_e = tk.Button(mod\_features, text="Add", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'),  
    command=lambda: chooseTypeOfQuestion(curr\_mod\_name))  
     
    add\_quest\_e.place(x=290, y=120)  
     
    mod\_del\_quest\_l = tk.Label(mod\_features, text="Delete / Edit a question from the list below: ",  
    fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    mod\_del\_quest\_l.place(x=10, y=195)  
     
    curr\_quest\_to\_del = tk.StringVar(mod\_features)  
    curr\_quest\_to\_del.set("Choose a Question") # default value  
     
    def del\_quest\_from\_db(quest\_name):  
    # Delete Feedback and BMA answers from db  
    len\_of\_quest = len(fetch\_all\_quest(curr\_mod\_name))  
    if len\_of\_quest == 1:  
    messagebox.showinfo("Error", "You cannot delete the existing question as the module will have 0 question to render."  
    " You must add another question first or delete the module itself.", parent=mod\_features)  
    return  
    q\_id = find\_quest\_id(quest\_name)  
    del\_feed\_quest\_from\_db(q\_id)  
    del\_bma\_quest\_from\_db(q\_id)  
    del\_quest\_execute(q\_id)  
    # Reset the List of Questions.  
    update\_option\_menu(curr\_quest\_to\_del\_m, curr\_mod\_name, curr\_quest\_to\_del)  
     
    curr\_quest\_to\_del\_m = tk.OptionMenu(mod\_features, curr\_quest\_to\_del, \*fetch\_all\_quest(curr\_mod\_name))  
    curr\_quest\_to\_del\_m.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    m\_q\_c = mod\_features.nametowidget(curr\_quest\_to\_del\_m.menuname)  
    m\_q\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_quest\_to\_del\_m.place(x=10, y=235)  
     
    mod\_del\_quest\_btn = tk.Button(mod\_features, text="Erase", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'),  
    command=lambda: del\_quest\_from\_db(curr\_quest\_to\_del.get()))  
    mod\_del\_quest\_btn.place(x=10, y=285)  
     
    mod\_edit\_quest\_btn = tk.Button(mod\_features, text="Edit", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'),  
    command=lambda: edit\_quest\_frame(curr\_mod\_name, curr\_quest\_to\_del.get())  
    )  
    mod\_edit\_quest\_btn.place(x=90, y=285)  
     
    # --------------- Change Module Name ----------------  
     
    def delBtnModsAndUpdate():  
    if self.isActive is True:  
    row = 2  
    col = 0  
    count = len(fetchModules())  
    while row < 7 and col < 2 and count > 0:  
    a = head.grid\_slaves(row, col)  
     
    if len(a) > 0:  
    a[0].destroy()  
    count -= 1  
    row += 1  
    if row == 7:  
    row = 2  
    col += 1  
    row = 2  
    col = 0  
    # We get all the modules and store them in the getModules  
    # Then we put them in columns of 5 rows while  
    # popping modules names from it so that no modules are repeated  
    getModules = fetchModules()  
     
    while row < 7 and col < 3 and len(getModules) > 0:  
    mod\_txt = getModules[-1][0]  
    mod\_btn = tk.Button(head, text=mod\_txt, fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), width=15, height=1,  
    command=lambda i=mod\_txt: change\_mod\_name(i))  
    mod\_btn.grid(row=row, column=col, padx=15, pady=20)  
     
    row += 1  
    if row == 7:  
    row = 2  
    col += 1  
    getModules.pop()  
     
    self.isActive = True  
     
    delBtnModsAndUpdate()  
     
    def toText(module):  
    # This additional function is used  
    # to separate the string module  
    # name from other non alphabet characters  
    i = 0  
    z = len(module) - 1  
    az = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'  
    while module[i] not in az:  
    i += 1  
    while module[z] not in az:  
    z -= 1  
     
    return module[i:z + 1]  
    def exist\_quest(quest\_name):  
    a = find\_quest\_id(quest\_name)  
    return False if a == [] else True  
     
    print(exist\_quest('sss'))  
    def isScoreValid(score):  
    alphabet = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-~`!@#$%^&\*()\_-+={}[]|\:;<,>.?/'  
    for x in score:  
    if x in alphabet:  
    return False  
    return True  
    # ---------------END OF USEFUL FUNCTIONS ---------------  
     
    # ---------------ADD MODULE---------------  
    def chooseTypeOfQuestion(curr\_mod\_name=False):  
    window = tk.Tk()  
    window.resizable(0, 0)  
    window.geometry("700x600")  
    head = tk.LabelFrame(window, text="Admin Page", fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'), bd=1)  
    head.pack(fill='both', expand='yes', padx=10, pady=10)  
     
    if curr\_mod\_name:  
    l1 = tk.Label(head, text="Select a type of question to add: True/False, MultipleChoice or BestMatch",  
    fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    l1.place(x=20, y=50)  
    else:  
    l1 = tk.Label(head,  
    text="You are creating a new module, so why don't create it along with an starting question?" +  
    " \n" + "Select a type of question to add: True/False, MultipleChoice or BestMatch",  
    fg="white", bg='#5D9DE5', font=('Helvetica', 12, 'bold'))  
    l1.place(x=20, y=50)  
     
    def TypeQuest(type, mod\_name=False):  
    # TRUE OR FALSE FORM  
     
    # 1.This frame collects all the information  
    # necessary to create the module and a  
    # first TF question in the database  
    #  
    # 2.It then add it to the database through  
    # a click an erase the window and update  
    # the buttons so the changes can me seen instantly.  
    #  
    # 3.In order to accomplish this,  
    # different type of sqlite  
    # commands are used, each with a different purpose  
     
    window.destroy()  
    tf\_form = tk.Tk()  
     
    tf\_form.resizable(0, 0)  
    tf\_form.geometry("700x600")  
     
    head = tk.LabelFrame(tf\_form, text="Admin Page", fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'), bd=1)  
    head.pack(fill='both', expand='yes', padx=20, pady=10)  
     
    l1 = tk.Label(head, text="Module Name: ", fg="white", bg='#5D9DE5', font=('Helvetica', 12, 'bold'))  
    l1.place(x=10, y=10)  
     
    max\_score = tk.Label(head,fg="white", bg='#5D9DE5', text="Score the user should get if answered correctly: ", font=('Helvetica', 13, 'bold'))  
    max\_score.place(x=10, y=60)  
     
    e\_score = tk.Entry(head, width=5, bd=5)  
    e\_score.place(x=410, y=60)  
     
    if mod\_name:  
    mod\_name\_title = tk.Label(head, text=curr\_mod\_name, fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    mod\_name\_title.place(x=350, y=10)  
    else:  
    e1 = tk.Entry(head, width=30, bd=5)  
    e1.place(x=130, y=10)  
     
    l2 = tk.Label(head, text="Question: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    l2.place(x=10, y=110)  
     
    e2 = tk.Entry(head, width=30, bd=5)  
    e2.place(x=100, y=110)  
     
     
    exist\_mod = True if mod\_name else False  
     
    def is\_mod\_name():  
    if mod\_name:  
    return mod\_name  
    return e1.get().lower()  
     
    if type == 'tf':  
    ans\_l = tk.Label(head, text="Correct Answer: ", fg="white", bg='#5D9DE5',font=('Helvetica', 11, 'bold'))  
    ans\_l.place(x=10, y=150)  
     
    ans\_e = tk.Entry(head, width=30, bd=5)  
    ans\_e.place(x=140, y=150)  
     
    def register\_tf():  
     
    if is\_mod\_name() == '':  
    messagebox.showinfo("Error", "Enter a module name", parent=tf\_form)  
    return  
    if e\_score.get() == '' or isScoreValid(e\_score.get()) is False or int(e\_score.get()) <= 0:  
    messagebox.showinfo("Error", "Make sure the score is not empty / is an integer / greater than 0", parent=tf\_form)  
    return  
     
    if e2.get() == '':  
    messagebox.showinfo("Error", "Question cannot be left empty ", parent=tf\_form)  
    return  
    if exist\_quest(e2.get()):  
    messagebox.showinfo("Error", "This question already exist. Please type another question.", parent=tf\_form)  
    return  
    if ans\_e.get().lower() == 'false' or ans\_e.get().lower() == 'true':  
    inc\_ans = 'true' if ans\_e.get().lower() == 'false' else 'false'  
     
    does\_mod\_exist\_in\_db = findModId(is\_mod\_name())  
     
    if does\_mod\_exist\_in\_db and mod\_name is False:  
    messagebox.showinfo("Error", "This module name already exists. "  
    "You will need to use another name.", parent=tf\_form)  
    return  
    # If it passes all the checks, Then you can create the module.  
    register\_mod\_DB(is\_mod\_name(),  
    e2.get(),  
    ans\_e.get().lower(),  
    inc\_ans,  
    tf\_form,  
    type,  
    e\_score.get(),  
    exist\_mod  
    )  
    else:  
    messagebox.showinfo("Error", "Please enter a false/true answer.", parent=tf\_form)  
    return  
     
    submit\_mod = tk.Button(head, text="add Module",  
    command=lambda: register\_tf())  
     
    submit\_mod.place(x=130, y=220)  
     
    if type == 'mcq':  
    # The user needs to able to choose how many answers/inc answers want to have  
    # therefore the approach I'm going to take is as follows  
    # 1. Let the user choose amount of answers  
    # 2. Let the user choose amount of inc\_ans  
    # 3. Base on that display entries for the user type on them.  
    # 4. Store ans and inc ans in DB  
    # 5. Prompt Feedback Frame  
    choices = [1, 2, 3]  
    l\_ans = tk.Label(head,  
    text="Right answers on the left. Wrong answers on the right. Max = 5", fg="white", bg='#5D9DE5',font=('Helvetica', 11, 'bold'))  
    l\_ans.place(x=10, y=150)  
     
    # Number of answers of the question  
    l\_num\_ans = tk.Label(head, text="N. Answers: ", font=('Helvetica', 13, 'bold'), fg="white", bg='#5D9DE5')  
    l\_num\_ans.place(x=10, y=180)  
     
    l\_num\_ans = tk.Label(head, text="N. Inc. Answers: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    l\_num\_ans.place(x=300, y=180)  
     
    ans\_1 = tk.Entry(head, width=30, bd=5)  
    ans\_1.place(x=100, y=220)  
     
    ans\_1 = tk.Entry(head, width=30, bd=5)  
    ans\_1.place(x=100, y=220)  
    ans\_2 = tk.Entry(head, width=30, bd=5)  
    ans\_2.place(x=100, y=250)  
     
    ans\_1 = tk.Entry(head, width=30, bd=5)  
    ans\_1.place(x=100, y=220)  
    ans\_2 = tk.Entry(head, width=30, bd=5)  
    ans\_2.place(x=100, y=250)  
    ans\_3 = tk.Entry(head, width=30, bd=5)  
    ans\_3.place(x=100, y=280)  
     
    inc\_ans\_1 = tk.Entry(head, width=30, bd=5)  
    inc\_ans\_1.place(x=300, y=220)  
     
    inc\_ans\_1 = tk.Entry(head, width=30, bd=5)  
    inc\_ans\_1.place(x=300, y=220)  
    inc\_ans\_2 = tk.Entry(head, width=30, bd=5)  
    inc\_ans\_2.place(x=300, y=250)  
     
    inc\_ans\_1 = tk.Entry(head, width=30, bd=5)  
    inc\_ans\_1.place(x=300, y=220)  
    inc\_ans\_2 = tk.Entry(head, width=30, bd=5)  
    inc\_ans\_2.place(x=300, y=250)  
    inc\_ans\_3 = tk.Entry(head, width=30, bd=5)  
    inc\_ans\_3.place(x=300, y=280)  
     
    def store\_inc\_and\_corr\_answers\_in\_db():  
    does\_mod\_exist\_in\_db = findModId(is\_mod\_name())  
    if is\_mod\_name() == '':  
    messagebox.showinfo("Error", "Enter a module name", parent=tf\_form)  
    return  
    if e\_score.get() == '' or isScoreValid(e\_score.get()) is False or int(e\_score.get()) <= 0:  
    messagebox.showinfo("Error", "Make sure the score is not empty / is an integer / greater than 0", parent=tf\_form)  
    return  
    if e2.get() == '':  
    messagebox.showinfo("Error", "Question cannot be left empty ", parent=tf\_form)  
    return  
    if exist\_quest(e2.get()):  
    messagebox.showinfo("Error", "This question already exist. Please type another question.", parent=tf\_form)  
    return  
    if does\_mod\_exist\_in\_db and mod\_name is False:  
    messagebox.showinfo("Error", "This module name already exists. "  
    "You will need to use another name.", parent=tf\_form)  
    return  
    is\_to\_much = [ans\_1.get(), ans\_2.get(), ans\_3.get(), inc\_ans\_3.get(), inc\_ans\_1.get(),  
    inc\_ans\_2.get()]  
    q = 0  
    for x in is\_to\_much:  
    if x != '':  
    q += 1  
    if q >= 6:  
    tk.messagebox.showinfo("showerror", "You can only add 5 answers/incorrect answers at most", parent=head)  
    return  
    canRun = False  
    def grab\_only\_ans(\*args):  
    l = []  
    c = 0  
    for x in args:  
    if x != '':  
    l.append(x)  
    for a in l:  
    if a == '':  
    c +=1  
    if c < 2:  
    messagebox.showinfo("Error", "You must add more than 1 answer/possible answer", parent=tf\_form)  
    return  
    canRun = True  
    return ",".join(l)  
     
    list\_of\_inc\_ans = grab\_only\_ans(inc\_ans\_1.get(), inc\_ans\_2.get(), inc\_ans\_3.get())  
    list\_of\_ans = grab\_only\_ans(ans\_1.get(), ans\_2.get(), ans\_3.get())  
     
    if canRun:  
    register\_mod\_DB(is\_mod\_name(),  
    e2.get(),  
    list\_of\_ans,  
    list\_of\_inc\_ans,  
    tf\_form,  
    type,  
    e\_score.get(),  
    exist\_mod)  
     
    submit\_mod = tk.Button(head, text="add Module", command=lambda: store\_inc\_and\_corr\_answers\_in\_db())  
    submit\_mod.place(x=130, y=320)  
     
    if type == 'bm':  
    # Enter 3-5 answers that others possible answers will be matched to  
     
    t\_ans = tk.Label(head,  
    text="Enter 1-5 answers. This will answers will be used as the match for other possible answers",  
    font=('Helvetica', 11, 'bold'), fg="white", bg='#5D9DE5')  
    t\_ans.place(x=10, y=150)  
     
    p\_ans\_1 = tk.Entry(head, width=30, bd=5)  
    p\_ans\_2 = tk.Entry(head, width=30, bd=5)  
    p\_ans\_3 = tk.Entry(head, width=30, bd=5)  
    p\_ans\_4 = tk.Entry(head, width=30, bd=5)  
    p\_ans\_5 = tk.Entry(head, width=30, bd=5)  
     
    p\_ans\_1.place(x=10, y=180)  
    p\_ans\_2.place(x=10, y=220)  
    p\_ans\_3.place(x=10, y=260)  
    p\_ans\_4.place(x=10, y=300)  
    p\_ans\_5.place(x=10, y=340)  
     
    def phase\_1\_of\_bma():  
    does\_mod\_exist\_in\_db = findModId(is\_mod\_name())  
    if is\_mod\_name() == '':  
    messagebox.showinfo("Error", "Enter a module name", parent=tf\_form)  
    return  
    if e\_score.get() == '' or isScoreValid(e\_score.get()) is False or int(e\_score.get()) <= 0 :  
    messagebox.showinfo("Error", "Make sure the score is not empty / is an integer / greater than 0", parent=tf\_form)  
    return  
    if e2.get() == '':  
    messagebox.showinfo("Error", "Question cannot be left empty ", parent=tf\_form)  
    return  
    if exist\_quest(e2.get()):  
    messagebox.showinfo("Error", "This question already exist. Please type another question.", parent=tf\_form)  
    return  
    if does\_mod\_exist\_in\_db and mod\_name is False:  
    messagebox.showinfo("Error", "This module name already exists. "  
    "You will need to use another name.", parent=tf\_form)  
    return  
    canExecute = False  
    def grab\_only\_ans(\*args):  
    l = []  
    for x in args:  
    if x != '':  
    l.append(x)  
    if l == []:  
    messagebox.showinfo("Error", "You must fill at least 1 field of answer/possible answer.", parent=tf\_form)  
    return  
    canExecute = True  
    return ",".join(l)  
    list\_of\_ans = grab\_only\_ans(p\_ans\_1.get(), p\_ans\_2.get(), p\_ans\_3.get(), p\_ans\_4.get(),  
    p\_ans\_5.get())  
    if canExecute:  
    register\_mod\_DB(  
    is\_mod\_name(),  
    e2.get(),  
    list\_of\_ans,  
    '',  
    tf\_form,  
    type,  
    e\_score.get(),  
    exist\_mod  
    )  
     
    submit\_mod\_bm = tk.Button(head, text="Continue", command=lambda: phase\_1\_of\_bma())  
    submit\_mod\_bm.place(x=200, y=370)  
     
    options = tk.LabelFrame(head, text="Options: ", font=('Helvetica', 18, 'bold'), fg="white", bg='#5D9DE5')  
    options.pack(fill='both', expand='yes', padx=20, pady=100)  
    op1 = tk.Button(options, text="TF",font=('Helvetica', 12, 'bold'), width=20, height=7,  
    command=lambda: TypeQuest('tf', curr\_mod\_name), fg="white", bg='#2B84E9')  
    op1.pack(side=tk.LEFT)  
    op2 = tk.Button(options, text="MCQ", font=('Helvetica', 12, 'bold'), width=20, height=7,  
    command=lambda: TypeQuest('mcq', curr\_mod\_name), fg="white", bg='#2B84E9', )  
    op2.pack(side=tk.LEFT)  
    op3 = tk.Button(options, text="BM", font=('Helvetica', 12, 'bold'), width=20, height=7,  
    command=lambda: TypeQuest('bm', curr\_mod\_name), fg="white", bg='#2B84E9')  
    op3.pack(side=tk.LEFT)  
     
    # BMA FRAME TO GET ALL POSSIBLE ANSWERS  
    def get\_inc\_ans\_from\_bma\_frame(quest, typeofQuest, ans, currForm):  
    currForm.destroy()  
    bma\_phase\_2 = tk.Tk()  
    bma\_phase\_2.resizable(0, 0)  
    bma\_phase\_2.geometry("700x600")  
     
    admin\_phase\_2 = tk.LabelFrame(bma\_phase\_2, text="Admin Page", fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'), bd=1)  
    admin\_phase\_2.pack(fill='both', expand='yes', padx=20, pady=10)  
     
    t\_phase\_2 = tk.Label(admin\_phase\_2,  
    text="Now, Type 1-5 possible answers and match them to their corresponding answer.",  
    fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    t\_phase\_2.place(x=10, y=10)  
     
    bma\_p2\_ans\_1 = tk.Entry(admin\_phase\_2, width=30, bd=5)  
    bma\_p2\_ans\_2 = tk.Entry(admin\_phase\_2, width=30, bd=5)  
    bma\_p2\_ans\_3 = tk.Entry(admin\_phase\_2, width=30, bd=5)  
    bma\_p2\_ans\_4 = tk.Entry(admin\_phase\_2, width=30, bd=5)  
    bma\_p2\_ans\_5 = tk.Entry(admin\_phase\_2, width=30, bd=5)  
     
    bma\_p2\_ans\_1.place(x=10, y=50)  
    bma\_p2\_ans\_2.place(x=10, y=100)  
    bma\_p2\_ans\_3.place(x=10, y=150)  
    bma\_p2\_ans\_4.place(x=10, y=200)  
    bma\_p2\_ans\_5.place(x=10, y=250)  
     
    # First Answer DropDown Menu list  
    curr\_bma\_ans\_1 = tk.StringVar(admin\_phase\_2)  
    curr\_bma\_ans\_1.set("Choose Match") # default value  
     
    curr\_bma\_ans\_1\_c = tk.OptionMenu(admin\_phase\_2, curr\_bma\_ans\_1, \*ans.split(','))  
    curr\_bma\_ans\_1\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    menu\_1 = admin\_phase\_2.nametowidget(curr\_bma\_ans\_1\_c.menuname)  
    menu\_1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_bma\_ans\_1\_c.place(x=200, y=45)  
     
    # Second Answer DropDown Menu List  
     
    curr\_bma\_ans\_2 = tk.StringVar(admin\_phase\_2)  
    curr\_bma\_ans\_2.set("Choose Match") # default value  
     
    curr\_bma\_ans\_2\_c = tk.OptionMenu(admin\_phase\_2, curr\_bma\_ans\_2, \*ans.split(','))  
    curr\_bma\_ans\_2\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    menu\_2 = admin\_phase\_2.nametowidget(curr\_bma\_ans\_2\_c.menuname)  
    menu\_2.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_bma\_ans\_2\_c.place(x=200, y=95)  
     
    # Third Answer Drop Down Menu List  
     
    curr\_bma\_ans\_3 = tk.StringVar(admin\_phase\_2)  
    curr\_bma\_ans\_3.set("Choose Match") # default value  
     
    curr\_bma\_ans\_3\_c = tk.OptionMenu(admin\_phase\_2, curr\_bma\_ans\_3, \*ans.split(','))  
    curr\_bma\_ans\_3\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    menu\_3 = admin\_phase\_2.nametowidget(curr\_bma\_ans\_3\_c.menuname)  
    menu\_3.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_bma\_ans\_3\_c.place(x=200, y=145)  
     
    # Fourth Answer Drop Down Menu List  
     
    curr\_bma\_ans\_4 = tk.StringVar(admin\_phase\_2)  
    curr\_bma\_ans\_4.set("Choose Match") # default value  
     
    curr\_bma\_ans\_4\_c = tk.OptionMenu(admin\_phase\_2, curr\_bma\_ans\_4, \*ans.split(','))  
    curr\_bma\_ans\_4\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    menu\_4 = admin\_phase\_2.nametowidget(curr\_bma\_ans\_4\_c.menuname)  
    menu\_4.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_bma\_ans\_4\_c.place(x=200, y=195)  
     
    # Fifth Answer Drop Down Menu list  
     
    curr\_bma\_ans\_5 = tk.StringVar(admin\_phase\_2)  
    curr\_bma\_ans\_5.set("Choose Match") # default value  
     
    curr\_bma\_ans\_5\_c = tk.OptionMenu(admin\_phase\_2, curr\_bma\_ans\_5, \*ans.split(','))  
    curr\_bma\_ans\_5\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    menu\_5 = admin\_phase\_2.nametowidget(curr\_bma\_ans\_5\_c.menuname)  
    menu\_5.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    curr\_bma\_ans\_5\_c.place(x=200, y=245)  
     
    def execute\_bma\_add():  
    q\_id, m\_id = find\_mod\_quest\_id(quest)  
    answers = []  
    if bma\_p2\_ans\_1.get() != '':  
    add\_bma\_ans(q\_id, curr\_bma\_ans\_1.get(), bma\_p2\_ans\_1.get(), m\_id)  
    answers.append(bma\_p2\_ans\_1.get())  
    if bma\_p2\_ans\_2.get() != '':  
    add\_bma\_ans(q\_id, curr\_bma\_ans\_2.get(), bma\_p2\_ans\_2.get(), m\_id)  
    answers.append(bma\_p2\_ans\_2.get())  
    if bma\_p2\_ans\_3.get() != '':  
    add\_bma\_ans(q\_id, curr\_bma\_ans\_3.get(), bma\_p2\_ans\_3.get(), m\_id)  
    answers.append(bma\_p2\_ans\_3.get())  
    if bma\_p2\_ans\_4.get() != '':  
    add\_bma\_ans(q\_id, curr\_bma\_ans\_4.get(), bma\_p2\_ans\_4.get(), m\_id)  
    answers.append(bma\_p2\_ans\_4.get())  
    if bma\_p2\_ans\_5.get() != '':  
    add\_bma\_ans(q\_id, curr\_bma\_ans\_5.get(), bma\_p2\_ans\_5.get(), m\_id)  
    answers.append(bma\_p2\_ans\_5.get())  
    update\_bms\_db\_poss\_ans(answers, q\_id)  
     
    add\_feed\_frame(q\_id, answers, bma\_phase\_2, m\_id, True)  
     
    bma\_btn = tk.Button(admin\_phase\_2, text="Continue", command=lambda: execute\_bma\_add())  
    bma\_btn.place(x=200, y=320)  
     
    def add\_feed\_frame(quest\_id, answers, prevForm, m\_id, bma=False):  
    if prevForm is not False:  
    prevForm.destroy()  
    feed\_form = tk.Tk()  
     
    fontFrame = tkFont.Font(  
    family="Arial",  
    size=16,  
    weight='bold')  
    ques\_title = tkFont.Font(  
    family="Arial",  
    size=14  
    )  
    feed\_form.resizable(0, 0)  
    feed\_form.geometry("700x600")  
     
    f\_head = tk.LabelFrame(feed\_form, text="Admin Page", fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'), bd=1)  
    f\_head.pack(fill='both', expand='yes', padx=20, pady=10)  
     
    if bma:  
    t1 = tk.Label(f\_head, text="Now write why do they match to each other", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    t1.place(x=10, y=10)  
    else:  
    t1 = tk.Label(f\_head, text="Now write why each answer is correct/incorrect:", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    t1.place(x=10, y=10)  
     
    l = len(answers)  
    c\_x, c\_y = 40, 50  
    # Hacky Way of displaying entries.  
    # Target: Display all answers with entries  
    # for the user to write feedback why it is wrong/right  
    # Since we cannot id entries and access them manually,  
    # Im going to first:  
    # 1. Find the length of answers  
    # 2. display As many entries as answers are.  
    # 3. base on length again insert feedback  
    t\_1 = tk.Label(f\_head, text="match to ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    t\_2 = tk.Label(f\_head, text="match to ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    t\_3 = tk.Label(f\_head, text="match to ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    t\_4 = tk.Label(f\_head, text="match to ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    t\_5 = tk.Label(f\_head, text="match to ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
     
    if bma:  
    fathers = get\_fathers\_from\_children(answers)  
    if l == 2:  
    t\_1.place(x=c\_x \* 5, y=c\_y)  
    t\_2.place(x=c\_x \* 5, y=c\_y \* 2)  
     
    f\_1 = tk.Label(f\_head, text=fathers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    f\_2 = tk.Label(f\_head, text=fathers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
     
    f\_1.place(x=c\_x \* 8, y=c\_y)  
    f\_2.place(x=c\_x \* 8, y=c\_y \* 2)  
    if l == 3:  
    t\_1.place(x=c\_x \* 5, y=c\_y)  
    t\_2.place(x=c\_x \* 5, y=c\_y \* 2)  
    t\_3.place(x=c\_x \* 5, y=c\_y \* 3)  
     
    f\_1 = tk.Label(f\_head, text=fathers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    f\_2 = tk.Label(f\_head, text=fathers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    f\_3 = tk.Label(f\_head, text=fathers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
     
    f\_1.place(x=c\_x \* 8, y=c\_y)  
    f\_2.place(x=c\_x \* 8, y=c\_y \* 2)  
    f\_3.place(x=c\_x \* 8, y=c\_y \* 3)  
     
    if l == 4:  
    t\_1.place(x=c\_x \* 5, y=c\_y)  
    t\_2.place(x=c\_x \* 5, y=c\_y \* 2)  
    t\_3.place(x=c\_x \* 5, y=c\_y \* 3)  
    t\_4.place(x=c\_x \* 5, y=c\_y \* 4)  
     
    f\_1 = tk.Label(f\_head, text=fathers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    f\_2 = tk.Label(f\_head, text=fathers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    f\_3 = tk.Label(f\_head, text=fathers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    f\_4 = tk.Label(f\_head, text=fathers[3], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
     
    f\_1.place(x=c\_x \* 8, y=c\_y)  
    f\_2.place(x=c\_x \* 8, y=c\_y \* 2)  
    f\_3.place(x=c\_x \* 8, y=c\_y \* 3)  
    f\_4.place(x=c\_x \* 8, y=c\_y \* 4)  
    if l == 5:  
    t\_1.place(x=c\_x \* 5, y=c\_y)  
    t\_2.place(x=c\_x \* 5, y=c\_y \* 2)  
    t\_3.place(x=c\_x \* 5, y=c\_y \* 3)  
    t\_4.place(x=c\_x \* 5, y=c\_y \* 4)  
    t\_5.place(x=c\_x \* 5, y=c\_y \* 5)  
     
    f\_1 = tk.Label(f\_head, text=fathers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    f\_2 = tk.Label(f\_head, text=fathers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    f\_3 = tk.Label(f\_head, text=fathers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    f\_4 = tk.Label(f\_head, text=fathers[3], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    f\_5 = tk.Label(f\_head, text=fathers[4], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
     
    f\_1.place(x=c\_x \* 8, y=c\_y)  
    f\_2.place(x=c\_x \* 8, y=c\_y \* 2)  
    f\_3.place(x=c\_x \* 8, y=c\_y \* 3)  
    f\_4.place(x=c\_x \* 8, y=c\_y \* 4)  
    f\_5.place(x=c\_x \* 8, y=c\_y \* 5)  
     
    if l == 2:  
    ans1 = tk.Label(f\_head, text=answers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans1.place(x=c\_x, y=c\_y)  
    e\_ans1 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans1.place(x=c\_x \* 11, y=c\_y)  
     
    ans2 = tk.Label(f\_head, text=answers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans2.place(x=c\_x, y=c\_y \* 2)  
    e\_ans2 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans2.place(x=c\_x \* 11, y=c\_y \* 2)  
    if l == 3:  
    ans1 = tk.Label(f\_head, text=answers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans1.place(x=c\_x, y=c\_y)  
    e\_ans1 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans1.place(x=c\_x \* 11, y=c\_y)  
     
    ans2 = tk.Label(f\_head, text=answers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans2.place(x=c\_x, y=c\_y \* 2)  
    e\_ans2 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans2.place(x=c\_x \* 11, y=c\_y \* 2)  
     
    ans3 = tk.Label(f\_head, text=answers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans3.place(x=c\_x, y=c\_y \* 3)  
    e\_ans3 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans3.place(x=c\_x \* 11, y=c\_y \* 3)  
     
    if l == 4:  
    ans1 = tk.Label(f\_head, text=answers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans1.place(x=c\_x, y=c\_y)  
    e\_ans1 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans1.place(x=c\_x \* 11, y=c\_y)  
     
    ans2 = tk.Label(f\_head, text=answers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans2.place(x=c\_x, y=c\_y \* 2)  
    e\_ans2 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans2.place(x=c\_x \* 11, y=c\_y \* 2)  
     
    ans3 = tk.Label(f\_head, text=answers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans3.place(x=c\_x, y=c\_y \* 3)  
    e\_ans3 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans3.place(x=c\_x \* 11, y=c\_y \* 3)  
     
    ans4 = tk.Label(f\_head, text=answers[3], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans4.place(x=c\_x, y=c\_y \* 4)  
    e\_ans4 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans4.place(x=c\_x \* 11, y=c\_y \* 4)  
    if l == 5:  
    ans1 = tk.Label(f\_head, text=answers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans1.place(x=c\_x, y=c\_y)  
    e\_ans1 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans1.place(x=c\_x \* 11, y=c\_y)  
     
    ans2 = tk.Label(f\_head, text=answers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans2.place(x=c\_x, y=c\_y \* 2)  
    e\_ans2 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans2.place(x=c\_x \* 11, y=c\_y \* 2)  
     
    ans3 = tk.Label(f\_head, text=answers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans3.place(x=c\_x, y=c\_y \* 3)  
    e\_ans3 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans3.place(x=c\_x \* 11, y=c\_y \* 3)  
     
    ans4 = tk.Label(f\_head, text=answers[3], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans4.place(x=c\_x, y=c\_y \* 4)  
    e\_ans4 = tk.Entry(f\_head, width=40, bd=5)  
    e\_ans4.place(x=c\_x \* 11, y=c\_y \* 4)  
     
    ans5 = tk.Label(f\_head, text=answers[4], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    ans5.place(x=c\_x, y=c\_y \* 5)  
    e\_ans5 = tk.Entry(f\_head, width=40,bd=5)  
    e\_ans5.place(x=c\_x \* 11, y=c\_y \* 5)  
     
    def insert\_all\_feed():  
    if l == 2:  
    if e\_ans1.get() == '' or e\_ans2.get() == '':  
    messagebox.showinfo("Error", "You must fill up both fields with their respective feedback.", parent=f\_head)  
    return  
    add\_feed\_to\_DB(quest\_id, answers[0], e\_ans1.get(), m\_id)  
    add\_feed\_to\_DB(quest\_id, answers[1], e\_ans2.get(), m\_id)  
    if l == 3:  
    if e\_ans1.get() == '' or e\_ans2.get() == '' or e\_ans3.get():  
    messagebox.showinfo("Error", "You must fill up both fields with their respective feedback.", parent=f\_head)  
    return  
    add\_feed\_to\_DB(quest\_id, answers[0], e\_ans1.get(), m\_id)  
    add\_feed\_to\_DB(quest\_id, answers[1], e\_ans2.get(), m\_id)  
    add\_feed\_to\_DB(quest\_id, answers[2], e\_ans3.get(), m\_id)  
    if l == 4:  
    if e\_ans1.get() == '' or e\_ans2.get() == '' or e\_ans3.get() or e\_ans4.get():  
    messagebox.showinfo("Error", "You must fill up both fields with their respective feedback.", parent=f\_head)  
    return  
    add\_feed\_to\_DB(quest\_id, answers[0], e\_ans1.get(), m\_id)  
    add\_feed\_to\_DB(quest\_id, answers[1], e\_ans2.get(), m\_id)  
    add\_feed\_to\_DB(quest\_id, answers[2], e\_ans3.get(), m\_id)  
    add\_feed\_to\_DB(quest\_id, answers[3], e\_ans4.get(), m\_id)  
    if l == 5:  
    if e\_ans1.get() == '' or e\_ans2.get() == '' or e\_ans3.get() or e\_ans4.get() or e\_ans5.get():  
    messagebox.showinfo("Error", "You must fill up both fields with their respective feedback.", parent=f\_head)  
    return  
    add\_feed\_to\_DB(quest\_id, answers[0], e\_ans1.get(), m\_id)  
    add\_feed\_to\_DB(quest\_id, answers[1], e\_ans2.get(), m\_id)  
    add\_feed\_to\_DB(quest\_id, answers[2], e\_ans3.get(), m\_id)  
    add\_feed\_to\_DB(quest\_id, answers[3], e\_ans4.get(), m\_id)  
    add\_feed\_to\_DB(quest\_id, answers[4], e\_ans5.get(), m\_id)  
     
    feed\_form.destroy()  
     
    feed\_form\_sub = tk.Button(f\_head, text="Submit", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), command=insert\_all\_feed)  
    feed\_form\_sub.place(x=500, y=500)  
     
    def register\_mod\_DB(mod\_name, start\_quest, ans, inc\_ans, currForm, typeOfQuestion, mark, mod\_exist=False):  
    #  
    # 1. We add the module to database so it has a Key  
    # 2. We find that key  
    # 3. We can now add questions to its own table having access to the specific foreign key (mod\_id)  
    #  
    # Add otherAnswer and times attribute when adding to DB  
     
    if typeOfQuestion == 'tf':  
    # ADD MODULE TO DB  
    onlyDeleteBtnModules()  
    if mod\_exist is False:  
    add\_mod(mod\_name)  
     
    e6 = findModId(mod\_name)  
    # ADD QUESTIONS TO DB  
    add\_quest(start\_quest, e6, inc\_ans, ans, mark, typeOfQuestion)  
     
    q\_id = find\_quest\_id(start\_quest)  
    delBtnModsAndUpdate()  
    # ADD FEEDBACK TO DB  
    all\_ans = [ans, inc\_ans]  
    # hacky way of inserting question id to the feedback  
    add\_feed\_frame(q\_id, all\_ans, currForm, e6)  
    elif typeOfQuestion == 'mcq':  
    all\_ans = inc\_ans.split(',') + ans.split(',')  
    onlyDeleteBtnModules()  
    if mod\_exist is False:  
    add\_mod(mod\_name)  
    e6 = findModId(mod\_name)  
    add\_quest(start\_quest, e6, inc\_ans, ans, mark, typeOfQuestion)  
    q\_id = find\_quest\_id(start\_quest)  
    delBtnModsAndUpdate()  
    add\_feed\_frame(q\_id, all\_ans, currForm, e6)  
    elif typeOfQuestion == 'bm':  
    # NEED TO CONSIDER TWO THINGS  
    # The Admin can add some options  
    # Each of these options can have some answers  
    # the app should store these answers and options  
    # How can we link these options and answer to determine which belongs to which?  
    # IDEA1: Create a new Answers Table that will hold the answers and its option.  
    # This way we can just fetch this entities and compare them with what the user matched in the app.  
    # Idea 2: ADD an "A" To the beginning of each option and Answer so we know they are connected.  
    # We would just need to check the first letter to determine the correctness of the user match.  
    onlyDeleteBtnModules()  
    if mod\_exist is False:  
    add\_mod(mod\_name)  
    e6 = findModId(mod\_name)  
    add\_quest(start\_quest, e6, inc\_ans, ans, mark, typeOfQuestion)  
    delBtnModsAndUpdate()  
    get\_inc\_ans\_from\_bma\_frame(start\_quest, typeOfQuestion, ans, currForm)  
     
    add\_mod\_btn = tk.Button(head, text="New Module",fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), command=chooseTypeOfQuestion)  
    add\_mod\_btn.grid(row=0, column=1, padx=(100, 0))  
     
    # -----------------ORDER TO DELETE A MODULE----------------  
    # 1. Delete All questions from Databases  
    # 2. Delete Module From database  
    # 3. Close window frame and DeleteAndUpdate  
    #  
    def delModuleFrame():  
    are\_there\_modules = fetchModules()  
    if len(are\_there\_modules) == 0:  
    messagebox.showinfo("Error", "There are not modules available to delete." + "\n" + "Add modules first.")  
    return  
    window = tk.Tk()  
    window.configure(bg='#5D9DE5')  
    window.resizable(0, 0)  
    window.geometry("500x200")  
    window.title("Delete Module")  
    currModule = tk.StringVar(window)  
    currModule.set("--Select Module--") # default value  
     
    # Here the dropdown menu is created  
    # using the 'data' modules  
    # from the database.  
    chooseTest = tk.OptionMenu(window, currModule, \*fetchModules())  
    chooseTest.config(fg="white", bg='#5D9DE5', font=('Helvetica', 11, 'bold'))  
    menu = window.nametowidget(chooseTest.menuname)  
    menu.config(fg="white", bg='#5D9DE5', font=('Helvetica', 11, 'bold'))  
    chooseTest.place(x=20, y=20)  
     
    def deleteModule():  
    # ----------------MAIN DELETE MODULE FUNCTION--------------  
     
    # In order to make the app more dynamic.  
    # 1. Remove all widget buttons from the frame  
    # 2. Delete all feedback from the db of feedback from module chosen id  
    # 2. Delete all questions from the db of questions from module chosen  
    # 3. Delete Module from DB  
    # 4. Display all buttons widgets again  
    # 5. Delete frame window itself  
    onlyDeleteBtnModules()  
    mod\_to\_delete = toText(currModule.get())  
    get\_mod\_id = findModId(mod\_to\_delete)  
    delAllFeedbackFromDB(get\_mod\_id)  
    delAllBmaFromDB(get\_mod\_id)  
    delAllQuestionsFromDB(get\_mod\_id)  
    delModFromDB(mod\_to\_delete)  
    delBtnModsAndUpdate()  
    window.destroy()  
     
    sub\_del = tk.Button(window, text="erase", command=deleteModule, fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    sub\_del.place(x=230, y=20)  
     
    alert\_txt = tk.Label(window, text="Careful! All questions & feedback will also be erased",  
    fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
    alert\_txt.place(x=20, y=120)  
     
    del\_mod\_btn = tk.Button(head, text="Del Module", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), command=delModuleFrame)  
    del\_mod\_btn.grid(row=0, column=2, padx=(10, 0))

## Class …

1. class QuizzApp(tk.Tk):  
    def \_\_init\_\_(self, \*args, \*\*kwargs):  
    tk.Tk.\_\_init\_\_(self, \*args, \*\*kwargs)  
    self.currScore = 0  
    #Create the window  
    window = tk.Frame(self)  
    window.pack()  
     
    window.grid\_rowconfigure(0, minsize=700)  
    window.grid\_columnconfigure(0, minsize=700)  
     
    """  
    This idea of maintaining the frames in a constant for loop came from this video.  
    https://www.youtube.com/watch?v=tpGjHRDEjCE&t=1153s&ab\_channel=IGTechTeam  
      
    Basically, it creates a dictionary to store all the classes of the questionnaire.  
    Then with the "change\_frame" function it changes to which class you want to visit.   
    """  
    self.containerOfFrames= {}  
    for f in (LogInPage, UserHomePage, AdminHomePage):  
     
    frame = f(window, self)  
    self.containerOfFrames[f] = frame  
    frame.grid(row=0, column=0, sticky="nsew")  
    self.change\_frame(AdminHomePage)  
     
    def change\_frame(self, page):  
    frame = self.containerOfFrames[page]  
    frame.tkraise()  
    """  
    This function can be used to get the score of the user at any point in time whenever a  
    quiz has been initialized.  
    There is got to be another function to update the score to 0.  
      
    """  
    def getScore(self):  
    print(self.currScore)

## Class …

1. class LogInPage(tk.Frame):  
    def \_\_init\_\_(self, parent, controller):  
    tk.Frame.\_\_init\_\_(self, parent)  
    # --------------------------LOGIN---------------------------------  
    fontBG = tkFont.Font(  
    family="Arial",  
    size=16,  
    weight='bold',  
    )  
     
    font\_small = tkFont.Font(  
    family="Arial",  
    size=12,  
    weight='bold',  
    )  
    """  
    The idea came from: https://www.youtube.com/watch?v=tpGjHRDEjCE&t=1153s&ab\_channel=IGTechTeam  
    I used part of the code from the video to develop the essential log in page that would  
    be the bridge between the user interface and admin interface.  
    I fully understand the small parts I replicated from the video.  
      
    """  
    border = tk.LabelFrame(self, text="Log In", bg='#02203c', fg="white", bd=1, font=fontBG)  
    border.pack(fill='both', expand='yes', padx=20, pady=150)  
     
    username = tk.Label(border, text="username", font=fontBG, fg="white", bg='#02203c')  
    username.place(x=50, y=20)  
     
    userInput = tk.Entry(border, width=30, bd=5)  
    userInput.place(x=180, y=20)  
     
    password = tk.Label(border, text="password", font=fontBG, fg="white", bg='#02203c')  
    password.place(x=50, y=80)  
     
    passInput = tk.Entry(border, show="\*", width=30, bd=5)  
    passInput.place(x=180, y=80)  
    testing = tk.Label(border, text="To enter the admin page use: admin as username and admin as password",  
    font=font\_small, fg="white", bg='#02203c')  
    testing.place(x=20, y=200)  
     
    # -------------------------------SUBMIT LOGIN--------------------------  
    def verify():  
    *"""  
    OPTION 1 ->  
    username and password match  
    in the database File for normal users  
    Should take them to the UI of normal users  
     
    """* with open("credential.txt", "r") as f:  
    # ["username, password", "username,password"]  
    info = f.readlines()  
    for user in info:  
     
    # u -> username, p -> password  
    # split them such that u -> "username" and p -> "password"  
    u, p = user.split(",")  
    # strip -> removes spaces at the end and beginning  
    # if u match our username input and p match our password input take user to next page  
    if u.strip() == userInput.get() and p.strip() == passInput.get():  
    controller.change\_frame(UserHomePage)  
    return  
    """  
    # OPTION 2 ->  
    # username and password match  
    # in the database file for ADMINS users  
    # Should take them to the UI for Admin Users  
    """  
    with open("AdminCredential.txt", "r") as f:  
    # ["username, password", "username,password"]  
    info = f.readlines()  
    for user in info:  
    # u -> username, p -> password  
    # split them such that u -> "username" and p -> "password"  
    u, p = user.split(",")  
    # strip -> removes spaces at the end and begining  
    # if u match our username input and p match our password input take user to next page  
    if u.strip() == userInput.get() and p.strip() == passInput.get():  
    controller.change\_frame(AdminHomePage)  
    return  
    messagebox.showinfo("Error", "Please provide a correct username and password")  
     
    # ---------------------REGISTRATION-----------------------------  
    submitBtn = tk.Button(border, text="Submit", command=verify, font=fontBG)  
    submitBtn.place(x=275, y=120)  
     
    def registerUser():  
    window = tk.Tk()  
     
    # make the window not resizable  
    window.resizable(0, 0)  
     
    window.title("Register")  
    l1 = tk.Label(window, text="Username: ", font=fontBG)  
    l1.place(x=10, y=10)  
     
    e1 = tk.Entry(window, width=30, bd=5)  
    e1.insert(tk.END, "username")  
    e1.place(x=200, y=10)  
     
    l2 = tk.Label(window, text="Password: ", font=fontBG)  
    l2.place(x=10, y=80)  
     
    e2 = tk.Entry(window, show="\*", width=30, bd=5)  
    e2.insert(tk.END, "password")  
    e2.place(x=200, y=80)  
     
    l3 = tk.Label(window, text="Confirm Password: ", font=fontBG)  
    l3.place(x=10, y=150)  
    e3 = tk.Entry(window, show="\*", width=30, bd=5)  
    e3.place(x=200, y=150)  
     
    """  
    This function open the database and  
    check whether the username of the user  
    is already in use returns True if it is,  
    otherwise False  
    """  
    def isNameUsed(name):  
    with open("credential.txt", 'r') as f:  
    info = f.readlines()  
    for user in info:  
    # u -> username, p -> password  
    # split them such that u -> "username" and p -> "password"  
    u, p = user.split(",")  
    # strip -> removes spaces at the end and beginning  
    # if u match our username input  
    # and p match our password input take user to next page  
    if u.strip() == name:  
    return True  
    return False  
     
    # ----------------------------submit registration --------------------------  
    def check():  
    isUsed = isNameUsed(e1.get())  
    if e1.get() != 'username' and e2.get() != "password" and isUsed is False:  
    if e2.get() == e3.get():  
    with open("credential.txt", "a") as f:  
    f.write(e1.get() + ',' + e2.get() + "\n")  
     
    messagebox.showinfo("Welcome", "You are now a fully registered")  
    else:  
    messagebox.showinfo("Error", "Your password didn't get match!")  
    else:  
    if isUsed:  
    messagebox.showinfo("Error", "This username is already in use")  
    return  
    messagebox.showinfo("Error", "Some field is missing, Please fill of all of them")  
     
    e4 = tk.Button(window, text="Sign In", command=check, font=fontBG)  
    e4.place(x=330, y=180)  
    window.geometry("480x250")  
     
    registerBtn = tk.Button(self, text="Register", bg='#02203c', font=fontBG, fg="white",  
    command=registerUser)  
    registerBtn.place(x=550, y=170)

## Class …

…

# Testing

*Describe the process you took to test your code and to make sure the program functions as required. Provide the detailed test plan used. Also, indicate the testing you did after integrating your code with your partner’s.*

*(COMPLETE THIS SECTION INDIVIDUALLY)*

Unfortunately, my partner did not deliver any functional feature so I could not integrate my code to her’s.

As the admin features developer, I had to interact with the database in multiple occasions to fetch, delete, update and add data. Therefore, I decided to use a white box technique to verify the flow of inputs and outputs through the questionnaire app before developing any feature. I used this technique every time I needed to add a new feature to the project which made the implementation simpler as I could see the steps from top to bottom.

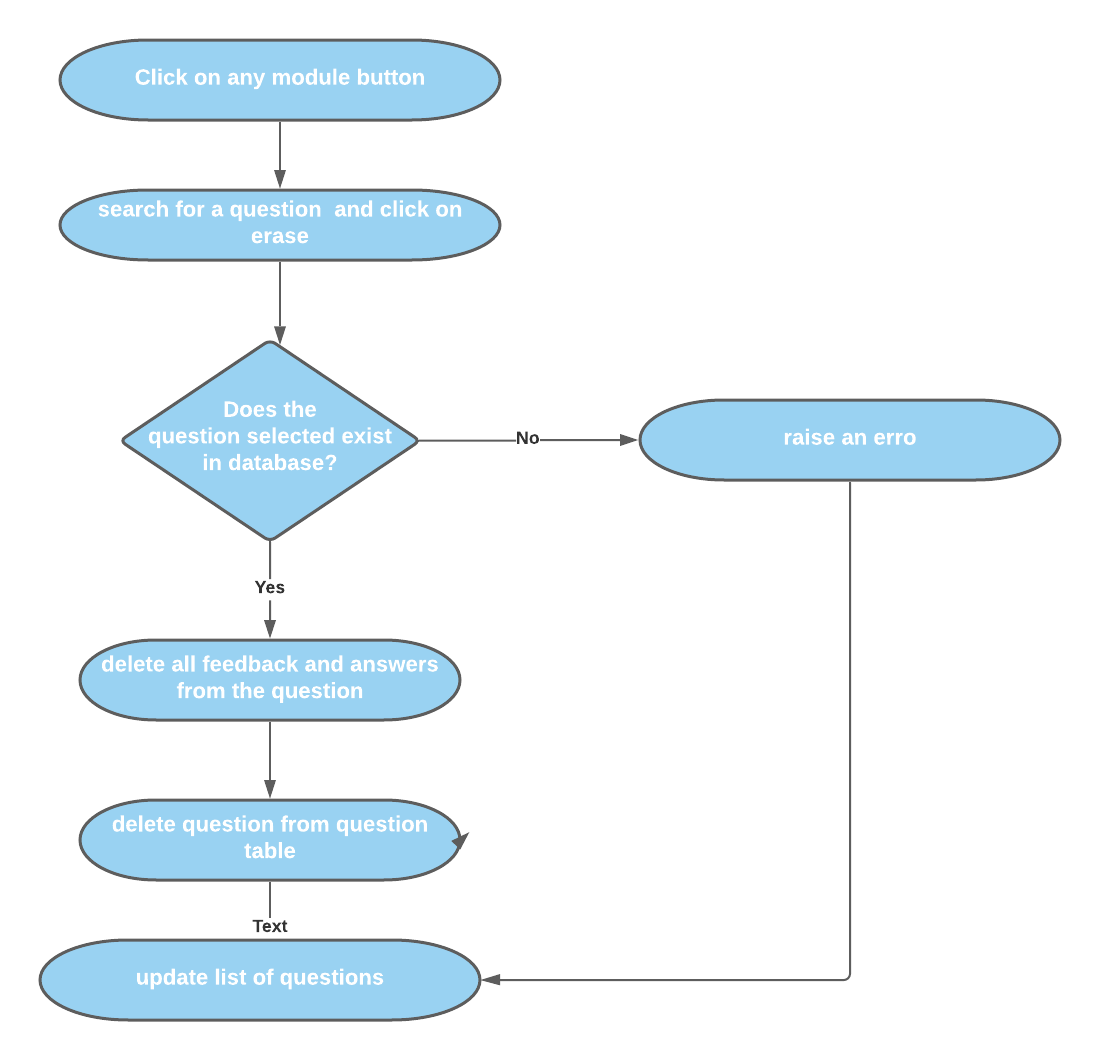
On some occasions multiple SQLite commands would be combined. For instance, when deleting a module, the program, first needs to delete all feedback and questions linked to the module and finally the module itself from the database. Having, the white box diagrams made this extremely easier to visualize and understand before starting to code.

Figure 2 white box, shows the process of deleting a question

See below some of the white box diagrams.

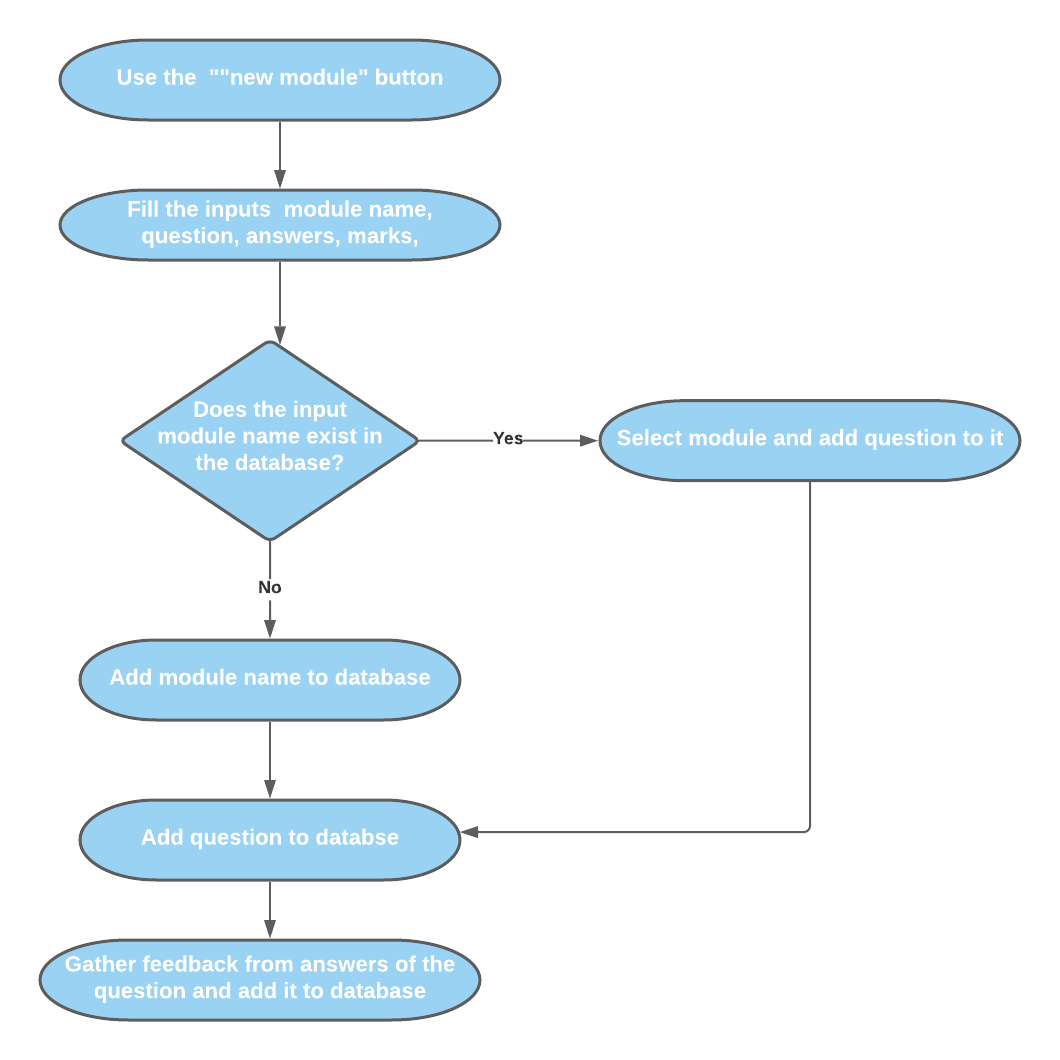


Figure 3 white box, shows the process of adding a new module

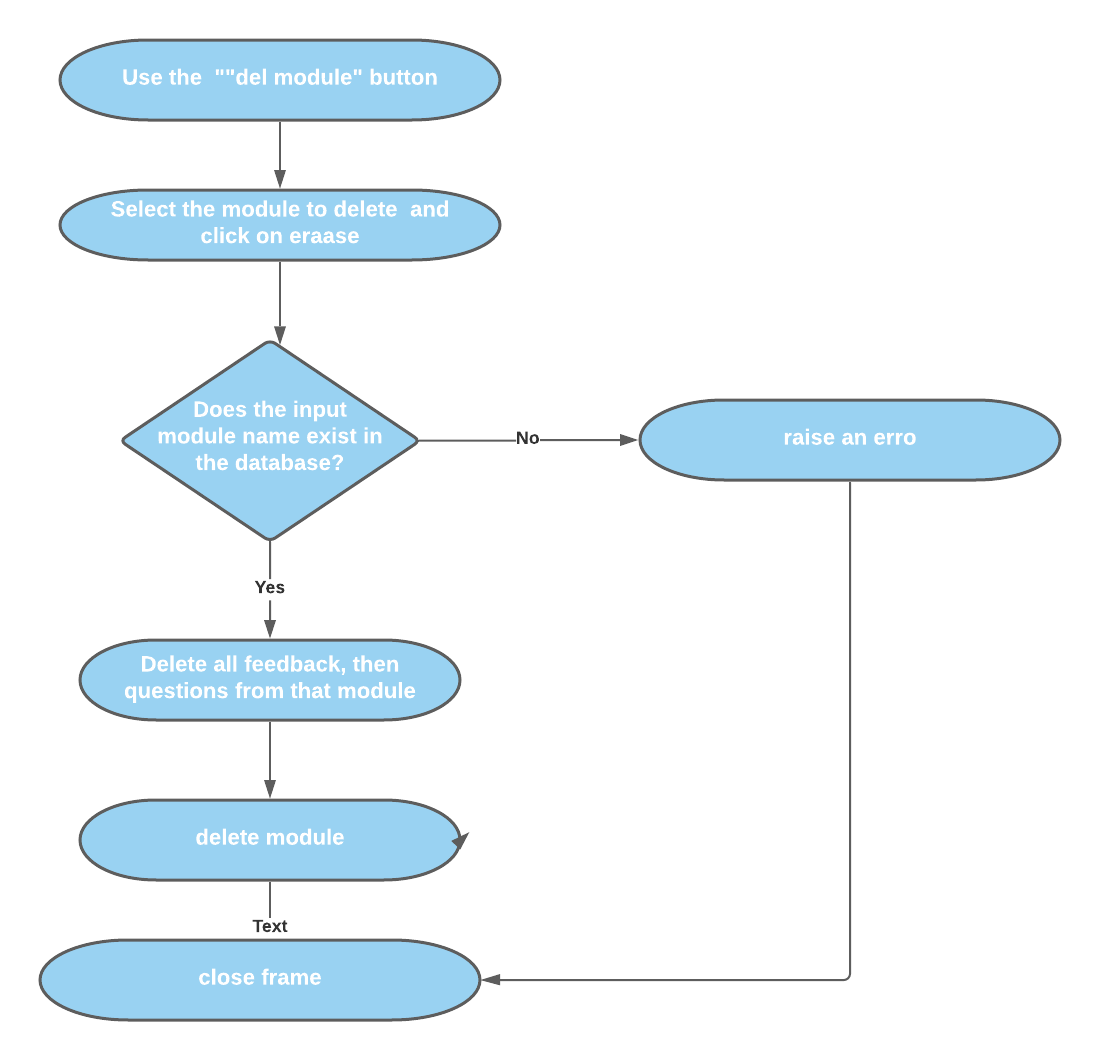


Figure 4 white box, shows the process of deleting a module

# Annotated Screenshots Demonstrating Implementation

*Provide screenshots that demonstrate the features implemented. Annotate each screenshot and if necessary, provide a brief description for* ***each*** *(****up to 100 words****) to explain the code in action. Make sure the screenshots make clear what you have implemented and achieved.*

*(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)*

## Feature F1 (Luis David Morales Huaygua)

## Sub-feature i- screenshots …

Chart, treemap chart

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generatedADD NEW MODULE TO THE QUIZ.

Figure Starting Question Type

Figure GUI of the Admin Page

Figure Buttons from the Admin Page

Table

Description automatically generatedGraphical user interface, application

Description automatically generated

Figure MODULE TABLE FROM THE DB

Figure MCQ TYPE FORM ON COMP.SCI MODULE

Graphical user interface, text, application, email

Description automatically generated

Figure QUESTION TABLE FROM THE DB

Graphical user interface, application

Description automatically generated

Figure FEEDBACK FRAME

Table

Description automatically generated

Figure FEEDBACK TABLE FROM THE DB

EDIT MODULE NAME DIRECTLY FROM THE GUI.

Graphical user interface

Description automatically generated

Figure COMP.SCI BUTTON CLICKED

Graphical user interface, text, application

Description automatically generated

Figure EDIT MODULE FRAME NEW MODULE ENTRY

**DELETE MODULE DIRECTLY FROM THE GUI.**

Graphical user interface, application

Description automatically generated

Figure DEL MODULE CLICKED AND SELECTION OF MODULE FROM LIST OF MODULES

## Sub-feature ii- screenshots …

ADD QUESTION AND POSSIBLE ANSWER.

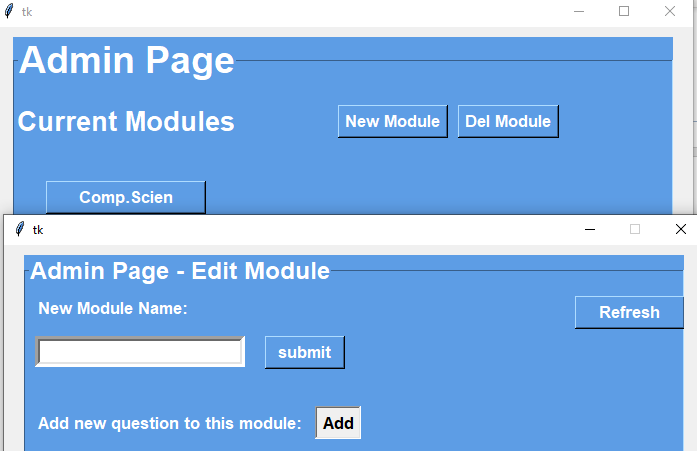


Figure ADD NEW QUESTION TO MODULE BUTTON CLICKED

Chart, treemap chart

Description automatically generated

Figure SELECT TYPE OF QUESTION FOR A NO NEW MODULE

Graphical user interface, application

Description automatically generated

Figure BM TYPE OF QUESTION FORM

Graphical user interface, application

Description automatically generatedGraphical user interface

Description automatically generated

Figure FEEDBACK FRAME

Figure BMA TYPE OF QUESTION FORM MATCHING THE WORDS

Graphical user interface, application

Description automatically generatedEDIT / DELETE AN EXISTING QUESTION.

Figure EDIT MODULE FRAME, CHOOSING A QUESTION TO DELETE/EDIT

Graphical user interface, text

Description automatically generated

Figure EDIT MOREDULE FRAME, EDIT BUTTON CLICKED

Graphical user interface, application

Description automatically generated

Figure EDIT QUESTION FRAME, SHOWING ALL THE INFORMATION FROM THE CHOSEN QUESTION.

## Sub-feature iii- screenshots …

Graphical user interface, application

Description automatically generatedLOGIN/MENU TO ACCESS ADMIN OR USER FEATURES.

Figure LOG IN MENU TO MAKE IMPLEMENATION OF CODE EASIER

## Feature F2 (Muthaliph, Gnei Tiara Rahiza)

## Sub-feature i- screenshots …

## Sub-feature ii- screenshots …

## Sub-feature iii- screenshots …

# Evaluation

*Give a reflective, critical self-evaluation of your experience developing the project and discuss what you would do if you had more time to work on the project. Answer the following questions for the reflection and write* ***350-400 words overall****. Please include an actual word count for this section.*

*(COMPLETE THIS SECTION INDIVIDUALLY)*

## Evaluate how well your design and implementation meet the requirements

This quiz has been tested for all sub-features of admin features and it passed all the test cases. However, due to my teammate’s performance, the questionnaire features could not be completed.

## Evaluate you own and your group’s performance

Although, my teammate and I met on some occasions to discuss the vision of the project I cannot say her performance reached my expectations. She was unable to implement a simple feature of the quiz by 3 days prior submission. Subsequently, she proceeds to block me on our main source of communication, presumably because of me asking her for updates on the project. As far as I am concerned, she did not ask for help, even though I was offering her my support in multiple occasions.

On the other hand, I was spending long hours building the admin feature and sending it to her whenever I update something so she can work on top of that. I managed to develop a fully functional Admin Frame and I believe it was because of my dedication and commitment on the project.

## What went well?

One of the things that I struggled the most was the interaction between the SQL file and python itself. The admin features required a lot of SQL manipulation because simply there were a lot of entities to be changed each time a button is clicked, literally. However, once I learned how to implement few of these queries, I found a way of doing things with less code. For instance, during the development of the “edit question” sub-feature I was doing a lot of calls to the database to update data, and therefore creating lot of functions for each of these calls. I made this simpler with a single function that would take the column we want to update and the new value as parameter.

## What went less well?

The implementation of more than 3 classes was not possible. This is because I believe that opening a new window and passing the necessary parameters to this new function window was easier than implementing a new class for each window I needed. This resulted in the F1 (my part) being less OOP oriented but overall, I am happy with the results.

## What was learnt?

During the development of the project, I learned a ton about best python practices, OOD and OOP which helped me improve my design skills when building projects overall. I also learned that it is very important to look at specifications of the problem firstly and tackle the main features before everything else. If you can build a solid base in your project, you can always come back to it if you need to re-do the project again.

## How would a similar task be completed differently?

A similar task could be done using different libraries/technologies for the GUI but using the same approach OOD/OOP.

## How could the module be improved?

During the development of my project, I faced many issues with my partner. She wasn’t doing particularly well delivering any feature, and I think that was because she wasn’t taking the project seriously. Therefore, I believe that by doing more progress checks during the term would be very helpful to each team and unstuck those who are facing challenges.

## Self-assessment

*Please assess yourself objectively for each section shown below and then enter the total mark you expect to get. Marks for each assessment criteria are indicated between parentheses.*

## Code development (70)

Features Implemented [30]

Sub-feature i (up to 8)

Sub-features have not been implemented – 0

Attempted, not complete or very buggy – 1 or 2

Implemented and functioning without errors but not integrated – 3 or 4

Implemented and fully integrated but buggy – 5 or 6

Implemented, fully integrated and functioning without errors – 7 or 8

Sub-feature ii (up to 10)

Sub-features have not been implemented – 0

Attempted, not complete or very buggy – 1 or 2

Implemented and functioning without errors but not integrated – 3 to 5

Implemented and fully integrated but buggy – 6 to 8

Implemented, fully integrated and functioning without errors – 9 or 10

Sub-feature iii (up to 12)

Sub-features has not been implemented – 0

Attempted, not complete or very buggy – 1 to 3

Implemented and functioning without errors but not integrated – 4 to 6

Implemented and fully integrated but buggy – 7 to 9

Implemented, fully integrated and functioning without errors – 10 to 12

**For this criterion I think I got: 30 out of 30**

Use of OOP techniques [25]

Abstraction (up to 10)

No classes have been created – 0

Classes have been created superficially and not instantiated or used – 1 or 2

Classes have been created but only some have been instantiated and used – 3 or 4

Useful classes and objects have been created and used correctly – 5 to 7

The use of classes and objects exceeds the specification – 8 to 10

Encapsulation (up to 10)

No encapsulation has been used – 0

Class variables and methods have been encapsulated superficially – 1 to 3

Class variables and methods have been encapsulated correctly – 4 to 6

The use of encapsulation exceeds the specification – 7 to 10

Inheritance (up to 5)

No inheritance has been used – 0

Classes have been inherited superficially – 1

Classes have been inherited correctly – 2 to 4

The use of inheritance exceeds the specification – 5

Bonus marks will be awarded for the appropriate use of polymorphism (bonus marks up to 10)

**For this criterion I think I got: 15 out of 25**

Quality of Code [15]

Code Duplication (up to 8)

Code contains too many unnecessary code repetition – 0

Regular occurrences of duplicate code – 1 to 3

Occasional duplicate code – 4 to 5

Very little duplicate code – 6 to 7

No duplicate code – 8

PEP8 Conventions and naming of variables, methods and classes (up to 4)

PEP8 and naming convention has not been used – 0

PEP8 and naming convention has been used occasionally – 1

PEP8 and naming convention has been used, but not regularly – 2

PEP8 and naming convention has been used regularly – 3

PEP8 convention used professionally and all items have been named correctly – 4

In-code Comments (up to 3)

No in-code comments – 0

Code contains occasional in-code comments – 1

Code contains useful and regular in-code comments – 2

Thoroughly commented, good use of docstrings, and header comments describing.py files – 3

**For this criterion I think I got: 12 out of 15**

## Documentation (20)

Design (up to 10) clear exposition about the design and decisions for OOP use

The documentation cannot be understood on first reading or mostly incomplete – 0

The documentation is readable, but a section(s) are missing – 1 to 3

The documentation is complete – 4 to 6

The documentation is complete and of a high standard – 7 to 10

Testing (5)

Testing has not been demonstrated in the documentation – 0

Little white box testing has been documented – 1 or 2

White box testing has been documented for all the coursework – 3 or 4

White box testing has been documented for the whole system – 5

Evaluation (5)

No evaluation was shown in the documentation – 0

The evaluation shows a lack of thought – 1 or 2

The evaluation shows thought – 3 or 4

The evaluation shows clear introspection, demonstrates increased awareness – 5

**For this criterion I think I got: 5 out of 20**

## Acceptance Tests - Demonstrations (10)

Final Demo (up to 10)

Not attended or no work demonstrated – 0

Work demonstrated was not up to the standard expected – 1 to 3

Work demonstrated was up to the standard expected – 4 to 7

Work demonstrated exceeded the standard expected – 8 to 10

**For this criterion I think I got: 8 out of 10**

**I think my overall mark would be: 70 out of 100**

# Group Pro forma

*Describe the division of work and agree percentage contributions. The pro forma must be signed by all group members and an identical copy provided in each report. If you cannot agree percentage contributions, please indicate so in the notes column and provide your reasoning.*

*(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)*

| **Partner ID** | **Tasks/Features Completed** | **%Contribution** | **Signature** | **Notes** |
| --- | --- | --- | --- | --- |
| **1** | Administration Features | 50% | Luis David Morales Huaygua | Sub-feature i.  Sub-feature ii.  Sub-feature iii. |
| **2** | Quiz Features | 0% |  | Sub-feature i.  Sub-feature ii.  Sub-feature iii. |
|  | **Total** | 50% |  |  |

# Appendix A: Code Listing

*Provide a complete listing of all the \*.py files in your PyCharm project. Make sure your code is well commented and applies professional Python convention (refer to* [*PEP 8*](https://www.python.org/dev/peps/pep-0008/) *for details). The code listed here must match that uploaded to Moodle. Please copy and paste the actual code – no screenshots please! You will lose marks if screenshots are provided instead of code.*

*(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)*

import tkinter as tk  
# from LogIn import LogInPage  
# from uHome import UserHomePage  
# from aHome import AdminHomePage  
from tkinter import messagebox  
import sqlite3  
import tkinter.font as tkFont  
  
import sys  
import os  
  
  
# HOW TO HANDLE THE ADMIN UI VS NORMAL USER UI?  
# Idea 1: create a database to store all  
# the modules with their respective questions  
# Then by default there will be 5 modules and 5 questions for each module.  
# Admin user have the power to MODIFY the database.  
# Example:  
# Admin user can delete/add questions  
# i. Delete:  
# admin will just need to click on a little button to the right of each question which will delete the question SOMEHOW  
# ii. ADD:  
# admin will click on the + sign and immediately a new window will pop up for the user to add the new question  
  
# admin user can delete/add modules  
# i. Delete:  
# admin will just click on the little button to the right of each module on the -Modules Frame- which will delete the module and its questions SOMEHOW.  
# ii. ADD:  
# admin will click on the + sign button and immediately a new window will pop up for the user to add the new module and 5 default questions.  
  
# Normal user select module  
  
# \*args store all arguments in tuple  
# \*\*kwargs store key value pairs.  
def findModId(module\_name):  
 conn = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = conn.execute("SELECT mod\_id FROM Modules where mod\_name = '" + module\_name + "';")  
 row = cursor.fetchall()  
  
 return str(row[0][0]) if row != [] else False  
  
  
def fetchModules():  
 modules = ''  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "SELECT mod\_name from Modules"  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 modules = count.fetchall()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to fetch data", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return modules  
  
  
def fetch\_all\_quest(curr\_mod\_name):  
 m\_id = findModId(curr\_mod\_name)  
 print(m\_id, 'module id')  
 questions = []  
 modules = ''  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "SELECT quest\_name from Questions " \  
 "where mod\_id = " + str(m\_id)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 modules = count.fetchall()  
  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to fetch questions", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
  
 for i in range(len(modules)):  
 questions.append(modules[i][0])  
  
 return questions  
  
  
def get\_fathers\_from\_children(answers):  
 fathers = []  
  
 for i in range(len(answers)):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "SELECT bma\_father from BestMatchAns " \  
 "where bma\_child = " + "'" + answers[i] + "'"  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 result = count.fetchone()[0]  
 fathers.append(result)  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to select father from child ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
  
 return fathers  
  
  
def find\_mod\_quest\_id(quest\_name):  
 result = []  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "SELECT quest\_id, mod\_id from Questions " \  
 "where quest\_name = " + "'" + quest\_name + "'"  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 result = count.fetchall()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to select quest and mod id ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
  
 return [int(result[0][0]), int(result[0][1])]  
  
  
def find\_quest\_id(quest\_name):  
 result = ''  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "SELECT quest\_id from Questions " \  
 "where quest\_name = " + "'" + quest\_name + "'"  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 result = count.fetchone()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to select quest id ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
  
 return result[0] if result is not None else []  
  
  
def find\_inc\_ans\_from\_quest(quest\_id):  
 result = ''  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "Select possible\_answers from Questions " \  
 "where quest\_id = '" + str(quest\_id) + "'"  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 result = count.fetchall()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to fetch all answers from question ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 ans = result[0][0]  
 return ans.split(',')  
  
  
def get\_father\_from\_child(child):  
 c = ''  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "SELECT bma\_father from BestMatchAns " \  
 "where bma\_child = " + "'" + child + "'"  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 result = count.fetchone()[0]  
 c = result  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to select father from child ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return c  
  
  
def find\_quest\_mark(quest\_id):  
 result = ''  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "SELECT quest\_mark from Questions " \  
 "where quest\_id = " + "'" + str(quest\_id) + "'"  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 result = count.fetchone()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to select quest id ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
  
 return result[0]  
  
  
def find\_ans\_from\_quest(quest\_id):  
 result = ''  
 ans = []  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "Select answer from Questions " \  
 "where quest\_id = '" + str(quest\_id) + "'"  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 result = count.fetchall()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to fetch all answers from question ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 for x in result:  
 ans.append(x[0])  
 return ans  
  
  
def add\_ans\_to\_bma(q\_id, father, child, m\_id):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "INSERT INTO BestMatchAns (" \  
 "quest\_id, bma\_father, bma\_child, mod\_id) " \  
 "Values ('" + str(q\_id) + "', '" + father + \  
 "', '" + child + "', '" + str(m\_id) + "')"  
  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed add ans to bma ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def del\_bma\_rows(q\_id):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "Delete from BestMatchAns where " \  
 "quest\_id = " + str(q\_id)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to fetch all answers from question ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def update\_quest(quest\_id, column, new\_answer, frame=False):  
 if new\_answer == '':  
 messagebox.showinfo("Error", "A field must have been left blank. Please fill the fields you edited.", parent=frame)  
 return  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "Update Questions " \  
 "Set " + column + " = '" + new\_answer + "' " \  
 "Where quest\_id = '" + str(quest\_id) + "'"  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to update answers in question ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def del\_feed\_quest\_from\_db(q\_id):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "Delete from Feedback " \  
 "Where quest\_id = " + str(q\_id)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 result = count.fetchone()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to delete feedback from db", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def del\_bma\_quest\_from\_db(q\_id):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "Delete from BestMatchAns " \  
 "Where quest\_id = " + str(q\_id)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 result = count.fetchone()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to delete bms answers from db ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def del\_quest\_execute(q\_id):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "Delete from Questions " \  
 "where quest\_id = " + str(q\_id)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 result = count.fetchone()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to delete question from db ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def find\_quest\_type(quest\_name):  
 result = ''  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "Select quest\_type from Questions " \  
 "where quest\_name = '" + quest\_name + "'"  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 result = count.fetchone()  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to fetch type of quest ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
  
 return result[0]  
  
  
def update\_mod\_name\_in\_db(curr\_name, desired\_name):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "Update Modules set mod\_name = " + \  
 "'" + desired\_name + "'" + \  
 " where mod\_name = " + "'" + \  
 curr\_name + "';"  
 print(sqlite\_insert\_query)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 print("record Updated")  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to Update Data Into Sqlite3", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def add\_feed\_to\_DB(q\_id, name, text, m\_id):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "INSERT INTO Feedback (quest\_id, feed\_ans\_name, feed\_text, mod\_id)\  
 Values " + "(" + str(q\_id) + ", '" + name + "' , '" \  
 + text + "'," + str(m\_id) + ");"  
 print(sqlite\_insert\_query)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 print("Feedback of Answer Succesfully Inserted")  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to insert feedback of answer into Sqlite ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def add\_mod(mod\_name):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "INSERT INTO Modules (mod\_name) Values " + "('" + mod\_name + "')"  
 print(sqlite\_insert\_query)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 print("Moduled added succesfully")  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to insert Module into sqlite table", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def add\_quest(quest\_name, quest\_mod\_id, possible\_answers, answer, mark, quest\_type):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
 sqlite\_insert\_query = "INSERT INTO Questions (quest\_name," \  
 " mod\_id, quest\_times, answer, possible\_answers, quest\_mark, quest\_type)" \  
 " Values " + "('" + quest\_name + "'" + ",'" + quest\_mod\_id + "'" + \  
 ", '" + str(0) + "'" + ",'" + answer + "'" + ",'" + possible\_answers + \  
 "', '" + mark + "', '" + quest\_type + "');"  
 print(sqlite\_insert\_query)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 print("Question added succesfully")  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to insert question into sqlite ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def add\_bma\_ans(quest\_id, father, child, mod\_id):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
 sqlite\_insert\_query = "INSERT INTO BestMatchAns (quest\_id," \  
 " bma\_father, bma\_child, mod\_id)" \  
 " Values " + "('" + str(quest\_id) + "'" + ",'" + father + "'" + \  
 ", '" + child + "'" + ",'" + str(mod\_id) + "');"  
 print(sqlite\_insert\_query)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 print("bma answer added succesfully")  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to insert bma answer into sqlite ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def update\_bms\_db\_poss\_ans(children, q\_id):  
 c = ",".join(children)  
  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
 sqlite\_insert\_query = "UPDATE QUESTIONS " \  
 "SET possible\_answers = " + "'" + c + "'" + " where quest\_id = " + str(q\_id)  
  
 print(sqlite\_insert\_query)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 print("bma possible answer updated succesfully")  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to insert bma possible answer child into sqlite ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def delAllFeedbackFromDB(moduleId):  
 # In order to delete a module,  
 # we first need to delete all  
 # the feedback & questions from that  
 # module because they are  
 # connected throught a  
 # foreign key and thus  
 # cannot be deleted without  
 # first all questions are removed.  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "DELETE FROM Feedback WHERE mod\_id = " + moduleId + ";"  
 print(sqlite\_insert\_query)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 print("Feedback Succesfully Deleted")  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to Delete Feedback into Sqlite ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def delAllBmaFromDB(moduleId):  
 # In order to delete a module,  
 # we first need to delete all  
 # the quesions & feedback & bma if it has from that  
 # module because they are  
 # connected throught a  
 # foreign key and thus  
 # cannot be deleted without  
 # first all questions are removed.  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "DELETE FROM BestMatchAns WHERE mod\_id = " + moduleId + ";"  
 print(sqlite\_insert\_query)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 print("bma answers Succesfully Deleted")  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to Delete bma answers ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def delAllQuestionsFromDB(moduleId):  
 # In order to delete a module,  
 # we first need to delete all  
 # the quesions & feedback from that  
 # module because they are  
 # connected throught a  
 # foreign key and thus  
 # cannot be deleted without  
 # first all questions are removed.  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Succesfully connected to SQLite")  
  
 sqlite\_insert\_query = "DELETE FROM Questions WHERE mod\_id = " + moduleId + ";"  
 print(sqlite\_insert\_query)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 print("Questions Succesfully Deleted")  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to Delete questions into Sqlite ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
def delModFromDB(moduleName):  
 try:  
 sqliteConnection = sqlite3.connect('./Databases/question\_bank.db')  
 cursor = sqliteConnection.cursor()  
 print("Successfully connected to SQLite")  
  
 sqlite\_insert\_query = "DELETE FROM Modules WHERE mod\_name = '" + moduleName + "';"  
 print(sqlite\_insert\_query)  
 count = cursor.execute(sqlite\_insert\_query)  
 sqliteConnection.commit()  
 print("Module Successfully Deleted")  
 cursor.close()  
 except sqlite3.Error as error:  
 print("Failed to Delete questions into Sqlite ", error)  
 finally:  
 if sqliteConnection:  
 sqliteConnection.close()  
 print("The SQLite connection is closed")  
 return  
  
  
class LogInPage(tk.Frame):  
 def \_\_init\_\_(self, parent, controller):  
 tk.Frame.\_\_init\_\_(self, parent)  
 # --------------------------LOGIN---------------------------------  
  
 """  
 The idea came from: https://www.youtube.com/watch?v=tpGjHRDEjCE&t=1153s&ab\_channel=IGTechTeam  
 I used part of the code from the video to develop the essential log in page that would  
 be the bridge between the user interface and admin interface.  
 I fully understand the small parts I replicated from the video.  
   
 """  
  
  
 border = tk.LabelFrame(self, text="Log In", fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'), bd=1)  
 border.pack(fill='both', expand='yes', padx=20, pady=150)  
  
 username = tk.Label(border, text="username", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 username.place(x=50, y=20)  
  
  
  
 exit\_l = tk.Button(self, text="Exit", fg="white", bg='#5D9DE5', font=('Helvetica', 20, 'bold'),  
 command=lambda: controller.exitSystem())  
 exit\_l.place(x=550, y=490)  
 userInput = tk.Entry(border, width=30, bd=5)  
 userInput.place(x=180, y=20)  
  
 password = tk.Label(border, text="password", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 password.place(x=50, y=80)  
  
 passInput = tk.Entry(border, show="\*", width=30, bd=5)  
 passInput.place(x=180, y=80)  
 enterAdmin= tk.Label(border, text="To enter the admin page use: admin as username and admin as password",  
 fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 enterAdmin.place(x=20, y=200)  
  
 enterUser = tk.Label(border, text="To enter the user page use: user as username and user as password",  
 fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 enterUser.place(x=20, y=250)  
  
 # -------------------------------SUBMIT LOGIN--------------------------  
 def verify():  
 *"""  
 OPTION 1 ->  
 username and password match  
 in the database File for normal users  
 Should take them to the UI of normal users  
  
 """* with open("credential.txt", "r") as f:  
 # ["username, password", "username,password"]  
 info = f.readlines()  
 for user in info:  
  
 # u -> username, p -> password  
 # split them such that u -> "username" and p -> "password"  
 u, p = user.split(",")  
 # strip -> removes spaces at the end and beginning  
 # if u match our username input and p match our password input take user to next page  
 if u.strip() == userInput.get() and p.strip() == passInput.get():  
 controller.change\_frame(UserHomePage)  
 return  
 """  
 # OPTION 2 ->  
 # username and password match  
 # in the database file for ADMINS users  
 # Should take them to the UI for Admin Users  
 """  
 with open("AdminCredential.txt", "r") as f:  
 # ["username, password", "username,password"]  
 info = f.readlines()  
 for user in info:  
 # u -> username, p -> password  
 # split them such that u -> "username" and p -> "password"  
 u, p = user.split(",")  
 # strip -> removes spaces at the end and begining  
 # if u match our username input and p match our password input take user to next page  
 if u.strip() == userInput.get() and p.strip() == passInput.get():  
 controller.change\_frame(AdminHomePage)  
 return  
 messagebox.showinfo("Error", "Please provide a correct username and password")  
  
 # ---------------------REGISTRATION-----------------------------  
 submitBtn = tk.Button(border, text="Submit", command=verify, fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 submitBtn.place(x=275, y=120)  
  
 def registerUser():  
 window = tk.Tk()  
  
 # make the window not resizable  
 window.resizable(0, 0)  
 window.configure( bg='#5D9DE5')  
 window.title("Register")  
 l1 = tk.Label(window, text="Username: ", fg="white", bg='#5D9DE5', font=('Helvetica', 11, 'bold'))  
 l1.place(x=10, y=10)  
  
 e1 = tk.Entry(window, width=30, bd=5)  
 e1.insert(tk.END, "username")  
 e1.place(x=200, y=10)  
  
 l2 = tk.Label(window, text="Password: ", fg="white", bg='#5D9DE5', font=('Helvetica', 11, 'bold'))  
 l2.place(x=10, y=80)  
  
 e2 = tk.Entry(window, show="\*", width=30, bd=5)  
 e2.insert(tk.END, "password")  
 e2.place(x=200, y=80)  
  
 l3 = tk.Label(window, text="Confirm Password: ", fg="white", bg='#5D9DE5', font=('Helvetica', 11, 'bold'))  
 l3.place(x=10, y=150)  
 e3 = tk.Entry(window, show="\*", width=30, bd=5)  
 e3.place(x=200, y=150)  
  
 """  
 This function open the database and  
 check whether the username of the user  
 is already in use returns True if it is,  
 otherwise False  
 """  
 def isNameUsed(name):  
 with open("credential.txt", 'r') as f:  
 info = f.readlines()  
 for user in info:  
 # u -> username, p -> password  
 # split them such that u -> "username" and p -> "password"  
 u, p = user.split(",")  
 # strip -> removes spaces at the end and beginning  
 # if u match our username input  
 # and p match our password input take user to next page  
 if u.strip() == name:  
 return True  
 return False  
  
 # ----------------------------submit registration --------------------------  
 def check():  
 isUsed = isNameUsed(e1.get())  
 if e1.get() != 'username' and e2.get() != "password" and isUsed is False:  
 if e2.get() == e3.get():  
 with open("credential.txt", "a") as f:  
 f.write(e1.get() + ',' + e2.get() + "\n")  
  
 messagebox.showinfo("Welcome", "You are now a fully registered")  
 else:  
 messagebox.showinfo("Error", "Your password didn't get match!")  
 else:  
 if isUsed:  
 messagebox.showinfo("Error", "This username is already in use")  
 return  
 messagebox.showinfo("Error", "Some field is missing, Please fill of all of them")  
  
 e4 = tk.Button(window, text="Sign In", command=check, fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 e4.place(x=330, y=180)  
 window.geometry("480x250")  
  
 registerBtn = tk.Button(self, text="Register", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'),  
 command=registerUser)  
 registerBtn.place(x=550, y=170)  
  
  
class UserHomePage(tk.Frame):  
 def \_\_init\_\_(self, parent, controller):  
 tk.Frame.\_\_init\_\_(self, parent)  
 # def score():  
 # return controller.getScore()  
 #  
 # def addScore():  
 # controller.currScore += 1  
 # button = tk.Button(self, text="click", command=score)  
 # button.grid(row=4, column=1)  
 #  
 # button = tk.Button(self, text="clickToAddScore", command=addScore)  
 # button.grid(row=5, column=1)  
 def closeSystem():  
 self.destroy()  
 exit\_l = tk.Button(self, text="Exit", fg="white", bg='#5D9DE5', font=('Helvetica', 20, 'bold'), command=lambda:closeSystem())  
 exit\_l.place(x=550, y=10)  
 for i in range(3):  
 self.columnconfigure(i, weight=1, minsize=75)  
 self.rowconfigure(i, weight=1, minsize=50)  
  
 for j in range(0, 3):  
 frame = tk.Frame(  
 master=self,  
 relief=tk.RAISED,  
 borderwidth=1  
 )  
 frame.grid(row=i, column=j, padx=5, pady=5)  
  
 label = tk.Label(master=frame, text=f"Row {i}\nColumn {j}")  
 label.pack(padx=5, pady=5)  
  
  
class AdminHomePage(tk.Frame):  
 def \_\_init\_\_(self, parent, controller):  
 tk.Frame.\_\_init\_\_(self, parent)  
 self.isActive = False  
  
  
 head = tk.LabelFrame(self, text="Admin Page", fg="white", bg='#5D9DE5', font=('Helvetica', 28, 'bold'), bd=1)  
 head.pack(fill='both', expand='yes', padx=20, pady=10)  
  
 titleLabel = tk.Label(head, text="Current Modules", fg="white", bg='#5D9DE5', font=('Helvetica', 20, 'bold'))  
 titleLabel.grid(row=0, column=0, pady=20)  
 def closeSystem():  
 self.destroy()  
 exit\_l = tk.Button(head, text="Exit", fg="white", bg='#5D9DE5', font=('Helvetica', 20, 'bold'), command=lambda:closeSystem())  
 exit\_l.place(x=550, y=540)  
  
 def update\_option\_menu(m\_quest\_to\_del, curr\_mod\_name, quest\_to\_del):  
 m = m\_quest\_to\_del['menu']  
 m.delete(0, 'end')  
 list\_of\_quest = fetch\_all\_quest(curr\_mod\_name)  
 for string in list\_of\_quest:  
 m.add\_command(label=string, command=lambda value=string: quest\_to\_del.set(value))  
  
 def onlyDeleteBtnModules():  
 row = 2  
 col = 0  
 count = len(fetchModules())  
 print(count, 'sss')  
 while row < 7 and col < 2 and count > 0:  
 a = head.grid\_slaves(row, col)  
 if len(a) > 0:  
 a[0].destroy()  
 count -= 1  
 row += 1  
 if row == 7:  
 row = 2  
 col += 1  
 self.isActive = False  
 return  
  
 # EDIT QUESTIONS MODE NAME  
 def edit\_quest\_frame(mod\_name, quest\_name):  
  
 window = tk.Tk()  
 window.resizable(0, 0)  
 window.geometry("700x600")  
 m\_quest\_features = tk.LabelFrame(window, text="Admin Page - Edit Question",fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'),  
 bd=1)  
 m\_quest\_features.pack(fill='both', expand='yes', padx=20, pady=10)  
  
 # Find type of quest and id:  
 type\_of\_q = find\_quest\_type(quest\_name)  
 quest\_id = find\_quest\_id(quest\_name)  
  
 quest\_module = tk.Label(m\_quest\_features, text="Module: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 quest\_module.place(x=10, y=10)  
  
 quest\_module\_title = tk.Label(m\_quest\_features, text=mod\_name, fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 quest\_module\_title.place(x=180, y=10)  
  
 quest\_title = tk.Label(m\_quest\_features, text="Question Name: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 quest\_title.place(x=10, y=60)  
  
 e\_quest\_title = tk.Entry(m\_quest\_features, width=30, bd=5)  
 e\_quest\_title.place(x=150, y=60)  
 e\_quest\_title.insert(tk.END, quest\_name)  
  
 quest\_t\_btn = tk.Button(m\_quest\_features,fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), text="change", command=lambda: update\_quest(  
 quest\_id,  
 'quest\_name',  
 e\_quest\_title.get(),  
 m\_quest\_features  
 ))  
 quest\_t\_btn.place(x=360, y=57)  
  
 if type\_of\_q == 'tf':  
 l\_ans = tk.Label(m\_quest\_features, text="Answer: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 l\_ans.place(x=10, y=100)  
 get\_ans\_tf = find\_ans\_from\_quest(quest\_id)  
 e\_ans\_tf = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_ans\_tf.place(x=10, y=130)  
 e\_ans\_tf.insert(tk.END, get\_ans\_tf)  
 # Update answer column  
 ans\_tf\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
 command=lambda: update\_quest(quest\_id, 'answer', e\_ans\_tf.get(), m\_quest\_features))  
 ans\_tf\_btn.place(x=10, y=160)  
  
 get\_inc\_ans\_tf = find\_inc\_ans\_from\_quest(quest\_id)  
 l\_inc\_ans = tk.Label(m\_quest\_features, text="Incorrect Answer: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 l\_inc\_ans.place(x=10, y=200)  
  
 e\_inc\_ans = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_inc\_ans.place(x=10, y=230)  
 e\_inc\_ans.insert(tk.END, get\_inc\_ans\_tf)  
 # Update possible answer column  
  
 ans\_tf\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
 command=lambda: update\_quest(quest\_id, 'possible\_answer', e\_inc\_ans.get(), m\_quest\_features))  
 ans\_tf\_btn.place(x=10, y=260)  
  
 # Update question Mark  
 get\_quest\_mark = find\_quest\_mark(quest\_id)  
 l\_quest\_mark = tk.Label(m\_quest\_features, text="Question Mark: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 l\_quest\_mark.place(x=10, y=290)  
 e\_quest\_mark = tk.Entry(m\_quest\_features, width=8, bd=5)  
 e\_quest\_mark.place(x=10, y=320)  
 e\_quest\_mark.insert(tk.END, get\_quest\_mark)  
  
 q\_m\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
 command=lambda: update\_quest(quest\_id, 'quest\_mark', e\_quest\_mark.get(), m\_quest\_features))  
 q\_m\_btn.place(x=10, y=350)  
 if type\_of\_q == 'mcq':  
 l\_ans = tk.Label(m\_quest\_features, text="Answer: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 l\_ans.place(x=10, y=100)  
 get\_ans = find\_ans\_from\_quest(quest\_id)  
 len\_of\_ans = len(get\_ans)  
 if len\_of\_ans == 1:  
 e\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_ans\_1.place(x=10, y=130)  
 e\_ans\_1.insert(tk.END, get\_ans[0])  
 elif len\_of\_ans == 2:  
 e\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_ans\_1.place(x=10, y=130)  
 e\_ans\_1.insert(tk.END, get\_ans[0])  
  
 e\_ans\_2 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_ans\_2.place(x=10, y=160)  
 e\_ans\_2.insert(tk.END, get\_ans[1])  
 elif len\_of\_ans == 3:  
 e\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_ans\_1.place(x=10, y=130)  
 e\_ans\_1.insert(tk.END, get\_ans[0])  
  
 e\_ans\_2 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_ans\_2.place(x=10, y=160)  
 e\_ans\_2.insert(tk.END, get\_ans[1])  
  
 e\_ans\_3 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_ans\_3.place(x=10, y=190)  
 e\_ans\_3.insert(tk.END, get\_ans[2])  
  
 get\_inc\_ans = find\_inc\_ans\_from\_quest(quest\_id)  
 l\_inc\_ans = tk.Label(m\_quest\_features, text="Incorrect Answers: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 l\_inc\_ans.place(x=10, y=250)  
 len\_of\_inc\_ans = len(get\_inc\_ans)  
  
 if len\_of\_inc\_ans == 1:  
 e\_inc\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_inc\_ans\_1.place(x=10, y=280)  
 e\_inc\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
 if len\_of\_inc\_ans == 2:  
 e\_inc\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_inc\_ans\_1.place(x=10, y=280)  
 e\_inc\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
  
 e\_inc\_ans\_2 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_inc\_ans\_2.place(x=10, y=310)  
 e\_inc\_ans\_2.insert(tk.END, get\_inc\_ans[1])  
  
 if len\_of\_inc\_ans == 3:  
 e\_inc\_ans\_1 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_inc\_ans\_1.place(x=10, y=280)  
 e\_inc\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
  
 e\_inc\_ans\_2 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_inc\_ans\_2.place(x=10, y=310)  
 e\_inc\_ans\_2.insert(tk.END, get\_inc\_ans[1])  
  
 e\_inc\_ans\_3 = tk.Entry(m\_quest\_features, width=20, bd=5)  
 e\_inc\_ans\_3.place(x=10, y=340)  
 e\_inc\_ans\_3.insert(tk.END, get\_inc\_ans[2])  
  
 def gather\_inc\_ans():  
 if len\_of\_inc\_ans == 1:  
 return ",".join([e\_inc\_ans\_1.get()])  
 if len\_of\_inc\_ans == 2:  
  
 return ",".join([e\_inc\_ans\_1.get(), e\_inc\_ans\_2.get()])  
 if len\_of\_inc\_ans == 3:  
 return ",".join([e\_inc\_ans\_1.get(), e\_inc\_ans\_2.get(), e\_inc\_ans\_3.get()])  
  
 def gather\_ans():  
 if len\_of\_ans == 1:  
 return ",".join([e\_ans\_1.get()])  
 if len\_of\_ans == 2:  
 return ",".join([e\_ans\_1.get(), e\_ans\_2.get()])  
 if len\_of\_ans == 3:  
 return ",".join([e\_ans\_1.get(), e\_ans\_2.get(), e\_ans\_3.get()])  
  
 ans\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
 command=lambda: update\_quest(quest\_id, 'answer', gather\_ans(), m\_quest\_features))  
 ans\_btn.place(x=10, y=220)  
  
 inc\_ans\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
 command=lambda: update\_quest(  
 quest\_id,  
 'possible\_answers',  
 gather\_inc\_ans(), m\_quest\_features))  
 inc\_ans\_btn.place(x=10, y=370)  
  
 get\_quest\_mark = find\_quest\_mark(quest\_id)  
 l\_quest\_mark = tk.Label(m\_quest\_features, text="Question Mark: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 l\_quest\_mark.place(x=10, y=410)  
 e\_quest\_mark = tk.Entry(m\_quest\_features, width=8, bd=5)  
 e\_quest\_mark.place(x=150, y=410)  
 e\_quest\_mark.insert(tk.END, get\_quest\_mark)  
  
 q\_m\_btn = tk.Button(m\_quest\_features, text="Apply", width=10,  
 command=lambda: update\_quest(quest\_id, 'quest\_mark', e\_quest\_mark.get(), m\_quest\_features))  
 q\_m\_btn.place(x=220, y=410)  
  
 if type\_of\_q == 'bm':  
 possible\_ans = tk.Label(m\_quest\_features, text="Possible Answers: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 possible\_ans.place(x=10, y=100)  
 get\_inc\_ans = find\_inc\_ans\_from\_quest(quest\_id)  
 get\_ans = ",".join(find\_ans\_from\_quest(quest\_id)).split(',')  
 l\_inc = len(get\_inc\_ans)  
  
 if l\_inc == 1:  
 e\_p\_ans\_1 = tk.Entry(m\_quest\_features, width=15, bd=5)  
 e\_p\_ans\_1.place(x=10, y=130)  
 e\_p\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
  
 bma\_e\_p\_ans1 = tk.StringVar(m\_quest\_features)  
 bma\_e\_p\_ans1.set(get\_father\_from\_child(get\_inc\_ans[0])) # default value  
  
 curr\_bma\_e\_p\_ans1 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans1, \*get\_ans)  
 curr\_bma\_e\_p\_ans1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_1 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans1.menuname)  
 curr\_1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_bma\_e\_p\_ans1.place(x=150, y=130)  
  
 if l\_inc == 2:  
 e\_p\_ans\_1 = tk.Entry(m\_quest\_features, width=15, bd=5)  
 e\_p\_ans\_1.place(x=10, y=130)  
 e\_p\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
  
 e\_p\_ans\_2 = tk.Entry(m\_quest\_features, width=15, bd=5)  
 e\_p\_ans\_2.place(x=10, y=180)  
 e\_p\_ans\_2.insert(tk.END, get\_inc\_ans[1])  
  
 bma\_e\_p\_ans1 = tk.StringVar(m\_quest\_features)  
 bma\_e\_p\_ans1.set(get\_father\_from\_child(get\_inc\_ans[0])) # default value  
  
 curr\_bma\_e\_p\_ans1 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans1, \*get\_ans)  
 curr\_bma\_e\_p\_ans1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_1 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans1.menuname)  
 curr\_1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_bma\_e\_p\_ans1.place(x=150, y=130)  
  
 bma\_e\_p\_ans2 = tk.StringVar(m\_quest\_features)  
 bma\_e\_p\_ans2.set(get\_father\_from\_child(get\_inc\_ans[1])) # default value  
  
 curr\_bma\_e\_p\_ans2 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans2, \*get\_ans)  
 curr\_bma\_e\_p\_ans2.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_2 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans2.menuname)  
 curr\_2.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_bma\_e\_p\_ans2.place(x=150, y=180)  
  
 if l\_inc == 3:  
  
 e\_p\_ans\_1 = tk.Entry(m\_quest\_features, width=15, bd=5)  
 e\_p\_ans\_1.place(x=10, y=130)  
 e\_p\_ans\_1.insert(tk.END, get\_inc\_ans[0])  
  
 e\_p\_ans\_2 = tk.Entry(m\_quest\_features, width=15, bd=5)  
 e\_p\_ans\_2.place(x=10, y=180)  
 e\_p\_ans\_2.insert(tk.END, get\_inc\_ans[1])  
  
 e\_p\_ans\_3 = tk.Entry(m\_quest\_features, width=15, bd=5)  
 e\_p\_ans\_3.place(x=10, y=230)  
 e\_p\_ans\_3.insert(tk.END, get\_inc\_ans[2])  
  
 bma\_e\_p\_ans1 = tk.StringVar(m\_quest\_features)  
 bma\_e\_p\_ans1.set(get\_father\_from\_child(get\_inc\_ans[0])) # default value  
  
 curr\_bma\_e\_p\_ans1 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans1, \*get\_ans)  
 curr\_bma\_e\_p\_ans1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_1 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans1.menuname)  
 curr\_1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_bma\_e\_p\_ans1.place(x=150, y=130)  
  
 bma\_e\_p\_ans2 = tk.StringVar(m\_quest\_features)  
 bma\_e\_p\_ans2.set(get\_father\_from\_child(get\_inc\_ans[1])) # default value  
  
 curr\_bma\_e\_p\_ans2 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans2, \*get\_ans)  
 curr\_bma\_e\_p\_ans2.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_2 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans2.menuname)  
 curr\_2.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_bma\_e\_p\_ans2.place(x=150, y=180)  
  
 bma\_e\_p\_ans3 = tk.StringVar(m\_quest\_features)  
 bma\_e\_p\_ans3.set(get\_father\_from\_child(get\_inc\_ans[2])) # default value  
  
 curr\_bma\_e\_p\_ans3 = tk.OptionMenu(m\_quest\_features, bma\_e\_p\_ans3, \*get\_ans)  
 curr\_bma\_e\_p\_ans3.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_3 = m\_quest\_features.nametowidget(curr\_bma\_e\_p\_ans3.menuname)  
 curr\_3.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_bma\_e\_p\_ans3.place(x=150, y=230)  
  
 def update\_bma\_ans():  
 p\_ans = []  
 ans = []  
 m\_id = findModId(mod\_name)  
  
 if l\_inc == 1:  
 p\_ans.append(e\_p\_ans\_1.get())  
 ans.append(bma\_e\_p\_ans1.get())  
 if l\_inc == 2:  
 p\_ans.append(e\_p\_ans\_1.get())  
 p\_ans.append(e\_p\_ans\_2.get())  
 ans.append(bma\_e\_p\_ans1.get())  
 ans.append(bma\_e\_p\_ans2.get())  
 if l\_inc == 3:  
 p\_ans.append(e\_p\_ans\_1.get())  
 p\_ans.append(e\_p\_ans\_2.get())  
 p\_ans.append(e\_p\_ans\_3.get())  
 ans.append(bma\_e\_p\_ans1.get())  
 ans.append(bma\_e\_p\_ans2.get())  
 ans.append(bma\_e\_p\_ans3.get())  
  
 i = 0  
  
 del\_bma\_rows(quest\_id)  
  
 while i < len(p\_ans):  
 add\_ans\_to\_bma(quest\_id, ans[i], p\_ans[i], m\_id)  
 i += 1  
 # return  
 del\_feed\_quest\_from\_db(quest\_id)  
 add\_feed\_frame(quest\_id, p\_ans, False, m\_id, True)  
 update\_quest(quest\_id, 'possible\_answers', ",".join(p\_ans), m\_quest\_features)  
  
 btn\_sub = tk.Button(m\_quest\_features, text="Submit", command=lambda:  
 update\_bma\_ans())  
 btn\_sub.place(x=10, y=260)  
  
 # Change mod name frame  
 def change\_mod\_name(curr\_mod\_name):  
 window = tk.Tk()  
 window.resizable(0, 0)  
 window.geometry("700x600")  
 mod\_features = tk.LabelFrame(window, text="Admin Page - Edit Module", fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'), bd=1)  
 mod\_features.pack(fill='both', expand='yes', padx=20, pady=10)  
  
 t\_new\_name = tk.Label(mod\_features, text="New Module Name: ", font=('Helvetica', 13, 'bold'), fg="white", bg='#5D9DE5')  
 t\_new\_name.place(x=10, y=10)  
 new\_name\_entry = tk.Entry(mod\_features, font=('Helvetica', 13, 'bold'), borderwidth=5, width=22, bd=5)  
 new\_name\_entry.place(x=10, y=50)  
 refresh\_btn = tk.Button(mod\_features, text="Refresh",fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), width=10,  
 command=lambda:  
 update\_option\_menu(curr\_quest\_to\_del\_m,  
 curr\_mod\_name,  
 curr\_quest\_to\_del  
 ))  
 refresh\_btn.place(x=550, y=10)  
  
 def execute\_mod\_change(curr\_name, desired\_name, frame=False):  
 if desired\_name == '':  
 messagebox.showinfo("Error",  
 "The field must have been left blank. Please fill the field you edited.",  
 parent=frame)  
 return  
 update\_mod\_name\_in\_db(curr\_name, desired\_name)  
 delBtnModsAndUpdate()  
 window.destroy()  
  
 new\_name\_sub = tk.Button(mod\_features, text="submit", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), width=7,  
 command=lambda: execute\_mod\_change(curr\_mod\_name, new\_name\_entry.get(), mod\_features))  
 new\_name\_sub.place(x=240, y=50)  
  
 # Add Question:  
  
 mod\_add\_quest\_l = tk.Label(mod\_features, text="Add new question to this module: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 mod\_add\_quest\_l.place(x=10, y=125)  
  
 add\_quest\_e = tk.Button(mod\_features, text="Add", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'),  
 command=lambda: chooseTypeOfQuestion(curr\_mod\_name))  
  
 add\_quest\_e.place(x=290, y=120)  
  
 mod\_del\_quest\_l = tk.Label(mod\_features, text="Delete / Edit a question from the list below: ",  
 fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 mod\_del\_quest\_l.place(x=10, y=195)  
  
 curr\_quest\_to\_del = tk.StringVar(mod\_features)  
 curr\_quest\_to\_del.set("Choose a Question") # default value  
  
 def del\_quest\_from\_db(quest\_name):  
 # Delete Feedback and BMA answers from db  
 len\_of\_quest = len(fetch\_all\_quest(curr\_mod\_name))  
 if len\_of\_quest == 1:  
 messagebox.showinfo("Error", "You cannot delete the existing question as the module will have 0 question to render."  
 " You must add another question first or delete the module itself.", parent=mod\_features)  
 return  
 q\_id = find\_quest\_id(quest\_name)  
 del\_feed\_quest\_from\_db(q\_id)  
 del\_bma\_quest\_from\_db(q\_id)  
 del\_quest\_execute(q\_id)  
 # Reset the List of Questions.  
 update\_option\_menu(curr\_quest\_to\_del\_m, curr\_mod\_name, curr\_quest\_to\_del)  
  
 curr\_quest\_to\_del\_m = tk.OptionMenu(mod\_features, curr\_quest\_to\_del, \*fetch\_all\_quest(curr\_mod\_name))  
 curr\_quest\_to\_del\_m.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 m\_q\_c = mod\_features.nametowidget(curr\_quest\_to\_del\_m.menuname)  
 m\_q\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_quest\_to\_del\_m.place(x=10, y=235)  
  
 mod\_del\_quest\_btn = tk.Button(mod\_features, text="Erase", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'),  
 command=lambda: del\_quest\_from\_db(curr\_quest\_to\_del.get()))  
 mod\_del\_quest\_btn.place(x=10, y=285)  
  
 mod\_edit\_quest\_btn = tk.Button(mod\_features, text="Edit", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'),  
 command=lambda: edit\_quest\_frame(curr\_mod\_name, curr\_quest\_to\_del.get())  
 )  
 mod\_edit\_quest\_btn.place(x=90, y=285)  
  
 # --------------- Change Module Name ----------------  
  
 def delBtnModsAndUpdate():  
 if self.isActive is True:  
 row = 2  
 col = 0  
 count = len(fetchModules())  
 while row < 7 and col < 2 and count > 0:  
 a = head.grid\_slaves(row, col)  
  
 if len(a) > 0:  
 a[0].destroy()  
 count -= 1  
 row += 1  
 if row == 7:  
 row = 2  
 col += 1  
 row = 2  
 col = 0  
 # We get all the modules and store them in the getModules  
 # Then we put them in columns of 5 rows while  
 # popping modules names from it so that no modules are repeated  
 getModules = fetchModules()  
  
 while row < 7 and col < 3 and len(getModules) > 0:  
 mod\_txt = getModules[-1][0]  
 mod\_btn = tk.Button(head, text=mod\_txt, fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), width=15, height=1,  
 command=lambda i=mod\_txt: change\_mod\_name(i))  
 mod\_btn.grid(row=row, column=col, padx=15, pady=20)  
  
 row += 1  
 if row == 7:  
 row = 2  
 col += 1  
 getModules.pop()  
  
 self.isActive = True  
  
 delBtnModsAndUpdate()  
  
 def toText(module):  
 # This additional function is used  
 # to separate the string module  
 # name from other non alphabet characters  
 i = 0  
 z = len(module) - 1  
 az = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'  
 while module[i] not in az:  
 i += 1  
 while module[z] not in az:  
 z -= 1  
  
 return module[i:z + 1]  
 def exist\_quest(quest\_name):  
 a = find\_quest\_id(quest\_name)  
 return False if a == [] else True  
  
 print(exist\_quest('sss'))  
 def isScoreValid(score):  
 alphabet = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-~`!@#$%^&\*()\_-+={}[]|\:;<,>.?/'  
 for x in score:  
 if x in alphabet:  
 return False  
 return True  
 # ---------------END OF USEFUL FUNCTIONS ---------------  
  
 # ---------------ADD MODULE---------------  
 def chooseTypeOfQuestion(curr\_mod\_name=False):  
 window = tk.Tk()  
 window.resizable(0, 0)  
 window.geometry("700x600")  
 head = tk.LabelFrame(window, text="Admin Page", fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'), bd=1)  
 head.pack(fill='both', expand='yes', padx=10, pady=10)  
  
 if curr\_mod\_name:  
 l1 = tk.Label(head, text="Select a type of question to add: True/False, MultipleChoice or BestMatch",  
 fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 l1.place(x=20, y=50)  
 else:  
 l1 = tk.Label(head,  
 text="You are creating a new module, so why don't create it along with an starting question?" +  
 " \n" + "Select a type of question to add: True/False, MultipleChoice or BestMatch",  
 fg="white", bg='#5D9DE5', font=('Helvetica', 12, 'bold'))  
 l1.place(x=20, y=50)  
  
 def TypeQuest(type, mod\_name=False):  
 # TRUE OR FALSE FORM  
  
 # 1.This frame collects all the information  
 # necessary to create the module and a  
 # first TF question in the database  
 #  
 # 2.It then add it to the database through  
 # a click an erase the window and update  
 # the buttons so the changes can me seen instantly.  
 #  
 # 3.In order to accomplish this,  
 # different type of sqlite  
 # commands are used, each with a different purpose  
  
 window.destroy()  
 tf\_form = tk.Tk()  
  
 tf\_form.resizable(0, 0)  
 tf\_form.geometry("700x600")  
  
 head = tk.LabelFrame(tf\_form, text="Admin Page", fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'), bd=1)  
 head.pack(fill='both', expand='yes', padx=20, pady=10)  
  
 l1 = tk.Label(head, text="Module Name: ", fg="white", bg='#5D9DE5', font=('Helvetica', 12, 'bold'))  
 l1.place(x=10, y=10)  
  
 max\_score = tk.Label(head,fg="white", bg='#5D9DE5', text="Score the user should get if answered correctly: ", font=('Helvetica', 13, 'bold'))  
 max\_score.place(x=10, y=60)  
  
 e\_score = tk.Entry(head, width=5, bd=5)  
 e\_score.place(x=410, y=60)  
  
 if mod\_name:  
 mod\_name\_title = tk.Label(head, text=curr\_mod\_name, fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 mod\_name\_title.place(x=350, y=10)  
 else:  
 e1 = tk.Entry(head, width=30, bd=5)  
 e1.place(x=130, y=10)  
  
 l2 = tk.Label(head, text="Question: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 l2.place(x=10, y=110)  
  
 e2 = tk.Entry(head, width=30, bd=5)  
 e2.place(x=100, y=110)  
  
  
 exist\_mod = True if mod\_name else False  
  
 def is\_mod\_name():  
 if mod\_name:  
 return mod\_name  
 return e1.get().lower()  
  
 if type == 'tf':  
 ans\_l = tk.Label(head, text="Correct Answer: ", fg="white", bg='#5D9DE5',font=('Helvetica', 11, 'bold'))  
 ans\_l.place(x=10, y=150)  
  
 ans\_e = tk.Entry(head, width=30, bd=5)  
 ans\_e.place(x=140, y=150)  
  
 def register\_tf():  
  
 if is\_mod\_name() == '':  
 messagebox.showinfo("Error", "Enter a module name", parent=tf\_form)  
 return  
 if e\_score.get() == '' or isScoreValid(e\_score.get()) is False or int(e\_score.get()) <= 0:  
 messagebox.showinfo("Error", "Make sure the score is not empty / is an integer / greater than 0", parent=tf\_form)  
 return  
  
 if e2.get() == '':  
 messagebox.showinfo("Error", "Question cannot be left empty ", parent=tf\_form)  
 return  
 if exist\_quest(e2.get()):  
 messagebox.showinfo("Error", "This question already exist. Please type another question.", parent=tf\_form)  
 return  
 if ans\_e.get().lower() == 'false' or ans\_e.get().lower() == 'true':  
 inc\_ans = 'true' if ans\_e.get().lower() == 'false' else 'false'  
  
 does\_mod\_exist\_in\_db = findModId(is\_mod\_name())  
  
 if does\_mod\_exist\_in\_db and mod\_name is False:  
 messagebox.showinfo("Error", "This module name already exists. "  
 "You will need to use another name.", parent=tf\_form)  
 return  
 # If it passes all the checks, Then you can create the module.  
 register\_mod\_DB(is\_mod\_name(),  
 e2.get(),  
 ans\_e.get().lower(),  
 inc\_ans,  
 tf\_form,  
 type,  
 e\_score.get(),  
 exist\_mod  
 )  
 else:  
 messagebox.showinfo("Error", "Please enter a false/true answer.", parent=tf\_form)  
 return  
  
 submit\_mod = tk.Button(head, text="add Module",  
 command=lambda: register\_tf())  
  
 submit\_mod.place(x=130, y=220)  
  
 if type == 'mcq':  
 # The user needs to able to choose how many answers/inc answers want to have  
 # therefore the approach I'm going to take is as follows  
 # 1. Let the user choose amount of answers  
 # 2. Let the user choose amount of inc\_ans  
 # 3. Base on that display entries for the user type on them.  
 # 4. Store ans and inc ans in DB  
 # 5. Prompt Feedback Frame  
 choices = [1, 2, 3]  
 l\_ans = tk.Label(head,  
 text="Right answers on the left. Wrong answers on the right. Max = 5", fg="white", bg='#5D9DE5',font=('Helvetica', 11, 'bold'))  
 l\_ans.place(x=10, y=150)  
  
 # Number of answers of the question  
 l\_num\_ans = tk.Label(head, text="N. Answers: ", font=('Helvetica', 13, 'bold'), fg="white", bg='#5D9DE5')  
 l\_num\_ans.place(x=10, y=180)  
  
 l\_num\_ans = tk.Label(head, text="N. Inc. Answers: ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 l\_num\_ans.place(x=300, y=180)  
  
 ans\_1 = tk.Entry(head, width=30, bd=5)  
 ans\_1.place(x=100, y=220)  
  
 ans\_1 = tk.Entry(head, width=30, bd=5)  
 ans\_1.place(x=100, y=220)  
 ans\_2 = tk.Entry(head, width=30, bd=5)  
 ans\_2.place(x=100, y=250)  
  
 ans\_1 = tk.Entry(head, width=30, bd=5)  
 ans\_1.place(x=100, y=220)  
 ans\_2 = tk.Entry(head, width=30, bd=5)  
 ans\_2.place(x=100, y=250)  
 ans\_3 = tk.Entry(head, width=30, bd=5)  
 ans\_3.place(x=100, y=280)  
  
 inc\_ans\_1 = tk.Entry(head, width=30, bd=5)  
 inc\_ans\_1.place(x=300, y=220)  
  
 inc\_ans\_1 = tk.Entry(head, width=30, bd=5)  
 inc\_ans\_1.place(x=300, y=220)  
 inc\_ans\_2 = tk.Entry(head, width=30, bd=5)  
 inc\_ans\_2.place(x=300, y=250)  
  
 inc\_ans\_1 = tk.Entry(head, width=30, bd=5)  
 inc\_ans\_1.place(x=300, y=220)  
 inc\_ans\_2 = tk.Entry(head, width=30, bd=5)  
 inc\_ans\_2.place(x=300, y=250)  
 inc\_ans\_3 = tk.Entry(head, width=30, bd=5)  
 inc\_ans\_3.place(x=300, y=280)  
  
 def store\_inc\_and\_corr\_answers\_in\_db():  
 does\_mod\_exist\_in\_db = findModId(is\_mod\_name())  
 if is\_mod\_name() == '':  
 messagebox.showinfo("Error", "Enter a module name", parent=tf\_form)  
 return  
 if e\_score.get() == '' or isScoreValid(e\_score.get()) is False or int(e\_score.get()) <= 0:  
 messagebox.showinfo("Error", "Make sure the score is not empty / is an integer / greater than 0", parent=tf\_form)  
 return  
 if e2.get() == '':  
 messagebox.showinfo("Error", "Question cannot be left empty ", parent=tf\_form)  
 return  
 if exist\_quest(e2.get()):  
 messagebox.showinfo("Error", "This question already exist. Please type another question.", parent=tf\_form)  
 return  
 if does\_mod\_exist\_in\_db and mod\_name is False:  
 messagebox.showinfo("Error", "This module name already exists. "  
 "You will need to use another name.", parent=tf\_form)  
 return  
 is\_to\_much = [ans\_1.get(), ans\_2.get(), ans\_3.get(), inc\_ans\_3.get(), inc\_ans\_1.get(),  
 inc\_ans\_2.get()]  
 q = 0  
 for x in is\_to\_much:  
 if x != '':  
 q += 1  
 if q >= 6:  
 tk.messagebox.showinfo("showerror", "You can only add 5 answers/incorrect answers at most", parent=head)  
 return  
 canRun = False  
 def grab\_only\_ans(\*args):  
 l = []  
 c = 0  
 for x in args:  
 if x != '':  
 l.append(x)  
 for a in l:  
 if a == '':  
 c +=1  
 if c < 2:  
 messagebox.showinfo("Error", "You must add more than 1 answer/possible answer", parent=tf\_form)  
 return  
 canRun = True  
 return ",".join(l)  
  
 list\_of\_inc\_ans = grab\_only\_ans(inc\_ans\_1.get(), inc\_ans\_2.get(), inc\_ans\_3.get())  
 list\_of\_ans = grab\_only\_ans(ans\_1.get(), ans\_2.get(), ans\_3.get())  
  
 if canRun:  
 register\_mod\_DB(is\_mod\_name(),  
 e2.get(),  
 list\_of\_ans,  
 list\_of\_inc\_ans,  
 tf\_form,  
 type,  
 e\_score.get(),  
 exist\_mod)  
  
 submit\_mod = tk.Button(head, text="add Module", command=lambda: store\_inc\_and\_corr\_answers\_in\_db())  
 submit\_mod.place(x=130, y=320)  
  
 if type == 'bm':  
 # Enter 3-5 answers that others possible answers will be matched to  
  
 t\_ans = tk.Label(head,  
 text="Enter 1-5 answers. This will answers will be used as the match for other possible answers",  
 font=('Helvetica', 11, 'bold'), fg="white", bg='#5D9DE5')  
 t\_ans.place(x=10, y=150)  
  
 p\_ans\_1 = tk.Entry(head, width=30, bd=5)  
 p\_ans\_2 = tk.Entry(head, width=30, bd=5)  
 p\_ans\_3 = tk.Entry(head, width=30, bd=5)  
 p\_ans\_4 = tk.Entry(head, width=30, bd=5)  
 p\_ans\_5 = tk.Entry(head, width=30, bd=5)  
  
 p\_ans\_1.place(x=10, y=180)  
 p\_ans\_2.place(x=10, y=220)  
 p\_ans\_3.place(x=10, y=260)  
 p\_ans\_4.place(x=10, y=300)  
 p\_ans\_5.place(x=10, y=340)  
  
 def phase\_1\_of\_bma():  
 does\_mod\_exist\_in\_db = findModId(is\_mod\_name())  
 if is\_mod\_name() == '':  
 messagebox.showinfo("Error", "Enter a module name", parent=tf\_form)  
 return  
 if e\_score.get() == '' or isScoreValid(e\_score.get()) is False or int(e\_score.get()) <= 0 :  
 messagebox.showinfo("Error", "Make sure the score is not empty / is an integer / greater than 0", parent=tf\_form)  
 return  
 if e2.get() == '':  
 messagebox.showinfo("Error", "Question cannot be left empty ", parent=tf\_form)  
 return  
 if exist\_quest(e2.get()):  
 messagebox.showinfo("Error", "This question already exist. Please type another question.", parent=tf\_form)  
 return  
 if does\_mod\_exist\_in\_db and mod\_name is False:  
 messagebox.showinfo("Error", "This module name already exists. "  
 "You will need to use another name.", parent=tf\_form)  
 return  
 canExecute = False  
 def grab\_only\_ans(\*args):  
 l = []  
 for x in args:  
 if x != '':  
 l.append(x)  
 if l == []:  
 messagebox.showinfo("Error", "You must fill at least 1 field of answer/possible answer.", parent=tf\_form)  
 return  
 canExecute = True  
 return ",".join(l)  
 list\_of\_ans = grab\_only\_ans(p\_ans\_1.get(), p\_ans\_2.get(), p\_ans\_3.get(), p\_ans\_4.get(),  
 p\_ans\_5.get())  
 if canExecute:  
 register\_mod\_DB(  
 is\_mod\_name(),  
 e2.get(),  
 list\_of\_ans,  
 '',  
 tf\_form,  
 type,  
 e\_score.get(),  
 exist\_mod  
 )  
  
 submit\_mod\_bm = tk.Button(head, text="Continue", command=lambda: phase\_1\_of\_bma())  
 submit\_mod\_bm.place(x=200, y=370)  
  
 options = tk.LabelFrame(head, text="Options: ", font=('Helvetica', 18, 'bold'), fg="white", bg='#5D9DE5')  
 options.pack(fill='both', expand='yes', padx=20, pady=100)  
 op1 = tk.Button(options, text="TF",font=('Helvetica', 12, 'bold'), width=20, height=7,  
 command=lambda: TypeQuest('tf', curr\_mod\_name), fg="white", bg='#2B84E9')  
 op1.pack(side=tk.LEFT)  
 op2 = tk.Button(options, text="MCQ", font=('Helvetica', 12, 'bold'), width=20, height=7,  
 command=lambda: TypeQuest('mcq', curr\_mod\_name), fg="white", bg='#2B84E9', )  
 op2.pack(side=tk.LEFT)  
 op3 = tk.Button(options, text="BM", font=('Helvetica', 12, 'bold'), width=20, height=7,  
 command=lambda: TypeQuest('bm', curr\_mod\_name), fg="white", bg='#2B84E9')  
 op3.pack(side=tk.LEFT)  
  
 # BMA FRAME TO GET ALL POSSIBLE ANSWERS  
 def get\_inc\_ans\_from\_bma\_frame(quest, typeofQuest, ans, currForm):  
 currForm.destroy()  
 bma\_phase\_2 = tk.Tk()  
 bma\_phase\_2.resizable(0, 0)  
 bma\_phase\_2.geometry("700x600")  
  
 admin\_phase\_2 = tk.LabelFrame(bma\_phase\_2, text="Admin Page", fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'), bd=1)  
 admin\_phase\_2.pack(fill='both', expand='yes', padx=20, pady=10)  
  
 t\_phase\_2 = tk.Label(admin\_phase\_2,  
 text="Now, Type 1-5 possible answers and match them to their corresponding answer.",  
 fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 t\_phase\_2.place(x=10, y=10)  
  
 bma\_p2\_ans\_1 = tk.Entry(admin\_phase\_2, width=30, bd=5)  
 bma\_p2\_ans\_2 = tk.Entry(admin\_phase\_2, width=30, bd=5)  
 bma\_p2\_ans\_3 = tk.Entry(admin\_phase\_2, width=30, bd=5)  
 bma\_p2\_ans\_4 = tk.Entry(admin\_phase\_2, width=30, bd=5)  
 bma\_p2\_ans\_5 = tk.Entry(admin\_phase\_2, width=30, bd=5)  
  
 bma\_p2\_ans\_1.place(x=10, y=50)  
 bma\_p2\_ans\_2.place(x=10, y=100)  
 bma\_p2\_ans\_3.place(x=10, y=150)  
 bma\_p2\_ans\_4.place(x=10, y=200)  
 bma\_p2\_ans\_5.place(x=10, y=250)  
  
 # First Answer DropDown Menu list  
 curr\_bma\_ans\_1 = tk.StringVar(admin\_phase\_2)  
 curr\_bma\_ans\_1.set("Choose Match") # default value  
  
 curr\_bma\_ans\_1\_c = tk.OptionMenu(admin\_phase\_2, curr\_bma\_ans\_1, \*ans.split(','))  
 curr\_bma\_ans\_1\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 menu\_1 = admin\_phase\_2.nametowidget(curr\_bma\_ans\_1\_c.menuname)  
 menu\_1.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_bma\_ans\_1\_c.place(x=200, y=45)  
  
 # Second Answer DropDown Menu List  
  
 curr\_bma\_ans\_2 = tk.StringVar(admin\_phase\_2)  
 curr\_bma\_ans\_2.set("Choose Match") # default value  
  
 curr\_bma\_ans\_2\_c = tk.OptionMenu(admin\_phase\_2, curr\_bma\_ans\_2, \*ans.split(','))  
 curr\_bma\_ans\_2\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 menu\_2 = admin\_phase\_2.nametowidget(curr\_bma\_ans\_2\_c.menuname)  
 menu\_2.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_bma\_ans\_2\_c.place(x=200, y=95)  
  
 # Third Answer Drop Down Menu List  
  
 curr\_bma\_ans\_3 = tk.StringVar(admin\_phase\_2)  
 curr\_bma\_ans\_3.set("Choose Match") # default value  
  
 curr\_bma\_ans\_3\_c = tk.OptionMenu(admin\_phase\_2, curr\_bma\_ans\_3, \*ans.split(','))  
 curr\_bma\_ans\_3\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 menu\_3 = admin\_phase\_2.nametowidget(curr\_bma\_ans\_3\_c.menuname)  
 menu\_3.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_bma\_ans\_3\_c.place(x=200, y=145)  
  
 # Fourth Answer Drop Down Menu List  
  
 curr\_bma\_ans\_4 = tk.StringVar(admin\_phase\_2)  
 curr\_bma\_ans\_4.set("Choose Match") # default value  
  
 curr\_bma\_ans\_4\_c = tk.OptionMenu(admin\_phase\_2, curr\_bma\_ans\_4, \*ans.split(','))  
 curr\_bma\_ans\_4\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 menu\_4 = admin\_phase\_2.nametowidget(curr\_bma\_ans\_4\_c.menuname)  
 menu\_4.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_bma\_ans\_4\_c.place(x=200, y=195)  
  
 # Fifth Answer Drop Down Menu list  
  
 curr\_bma\_ans\_5 = tk.StringVar(admin\_phase\_2)  
 curr\_bma\_ans\_5.set("Choose Match") # default value  
  
 curr\_bma\_ans\_5\_c = tk.OptionMenu(admin\_phase\_2, curr\_bma\_ans\_5, \*ans.split(','))  
 curr\_bma\_ans\_5\_c.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 menu\_5 = admin\_phase\_2.nametowidget(curr\_bma\_ans\_5\_c.menuname)  
 menu\_5.config(fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 curr\_bma\_ans\_5\_c.place(x=200, y=245)  
  
 def execute\_bma\_add():  
 q\_id, m\_id = find\_mod\_quest\_id(quest)  
 answers = []  
 if bma\_p2\_ans\_1.get() != '':  
 add\_bma\_ans(q\_id, curr\_bma\_ans\_1.get(), bma\_p2\_ans\_1.get(), m\_id)  
 answers.append(bma\_p2\_ans\_1.get())  
 if bma\_p2\_ans\_2.get() != '':  
 add\_bma\_ans(q\_id, curr\_bma\_ans\_2.get(), bma\_p2\_ans\_2.get(), m\_id)  
 answers.append(bma\_p2\_ans\_2.get())  
 if bma\_p2\_ans\_3.get() != '':  
 add\_bma\_ans(q\_id, curr\_bma\_ans\_3.get(), bma\_p2\_ans\_3.get(), m\_id)  
 answers.append(bma\_p2\_ans\_3.get())  
 if bma\_p2\_ans\_4.get() != '':  
 add\_bma\_ans(q\_id, curr\_bma\_ans\_4.get(), bma\_p2\_ans\_4.get(), m\_id)  
 answers.append(bma\_p2\_ans\_4.get())  
 if bma\_p2\_ans\_5.get() != '':  
 add\_bma\_ans(q\_id, curr\_bma\_ans\_5.get(), bma\_p2\_ans\_5.get(), m\_id)  
 answers.append(bma\_p2\_ans\_5.get())  
 update\_bms\_db\_poss\_ans(answers, q\_id)  
  
 add\_feed\_frame(q\_id, answers, bma\_phase\_2, m\_id, True)  
  
 bma\_btn = tk.Button(admin\_phase\_2, text="Continue", command=lambda: execute\_bma\_add())  
 bma\_btn.place(x=200, y=320)  
  
 def add\_feed\_frame(quest\_id, answers, prevForm, m\_id, bma=False):  
 if prevForm is not False:  
 prevForm.destroy()  
 feed\_form = tk.Tk()  
  
 fontFrame = tkFont.Font(  
 family="Arial",  
 size=16,  
 weight='bold')  
 ques\_title = tkFont.Font(  
 family="Arial",  
 size=14  
 )  
 feed\_form.resizable(0, 0)  
 feed\_form.geometry("700x600")  
  
 f\_head = tk.LabelFrame(feed\_form, text="Admin Page", fg="white", bg='#5D9DE5', font=('Helvetica', 18, 'bold'), bd=1)  
 f\_head.pack(fill='both', expand='yes', padx=20, pady=10)  
  
 if bma:  
 t1 = tk.Label(f\_head, text="Now write why do they match to each other", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 t1.place(x=10, y=10)  
 else:  
 t1 = tk.Label(f\_head, text="Now write why each answer is correct/incorrect:", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 t1.place(x=10, y=10)  
  
 l = len(answers)  
 c\_x, c\_y = 40, 50  
 # Hacky Way of displaying entries.  
 # Target: Display all answers with entries  
 # for the user to write feedback why it is wrong/right  
 # Since we cannot id entries and access them manually,  
 # Im going to first:  
 # 1. Find the length of answers  
 # 2. display As many entries as answers are.  
 # 3. base on length again insert feedback  
 t\_1 = tk.Label(f\_head, text="match to ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 t\_2 = tk.Label(f\_head, text="match to ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 t\_3 = tk.Label(f\_head, text="match to ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 t\_4 = tk.Label(f\_head, text="match to ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 t\_5 = tk.Label(f\_head, text="match to ", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
  
 if bma:  
 fathers = get\_fathers\_from\_children(answers)  
 if l == 2:  
 t\_1.place(x=c\_x \* 5, y=c\_y)  
 t\_2.place(x=c\_x \* 5, y=c\_y \* 2)  
  
 f\_1 = tk.Label(f\_head, text=fathers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 f\_2 = tk.Label(f\_head, text=fathers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
  
 f\_1.place(x=c\_x \* 8, y=c\_y)  
 f\_2.place(x=c\_x \* 8, y=c\_y \* 2)  
 if l == 3:  
 t\_1.place(x=c\_x \* 5, y=c\_y)  
 t\_2.place(x=c\_x \* 5, y=c\_y \* 2)  
 t\_3.place(x=c\_x \* 5, y=c\_y \* 3)  
  
 f\_1 = tk.Label(f\_head, text=fathers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 f\_2 = tk.Label(f\_head, text=fathers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 f\_3 = tk.Label(f\_head, text=fathers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
  
 f\_1.place(x=c\_x \* 8, y=c\_y)  
 f\_2.place(x=c\_x \* 8, y=c\_y \* 2)  
 f\_3.place(x=c\_x \* 8, y=c\_y \* 3)  
  
 if l == 4:  
 t\_1.place(x=c\_x \* 5, y=c\_y)  
 t\_2.place(x=c\_x \* 5, y=c\_y \* 2)  
 t\_3.place(x=c\_x \* 5, y=c\_y \* 3)  
 t\_4.place(x=c\_x \* 5, y=c\_y \* 4)  
  
 f\_1 = tk.Label(f\_head, text=fathers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 f\_2 = tk.Label(f\_head, text=fathers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 f\_3 = tk.Label(f\_head, text=fathers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 f\_4 = tk.Label(f\_head, text=fathers[3], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
  
 f\_1.place(x=c\_x \* 8, y=c\_y)  
 f\_2.place(x=c\_x \* 8, y=c\_y \* 2)  
 f\_3.place(x=c\_x \* 8, y=c\_y \* 3)  
 f\_4.place(x=c\_x \* 8, y=c\_y \* 4)  
 if l == 5:  
 t\_1.place(x=c\_x \* 5, y=c\_y)  
 t\_2.place(x=c\_x \* 5, y=c\_y \* 2)  
 t\_3.place(x=c\_x \* 5, y=c\_y \* 3)  
 t\_4.place(x=c\_x \* 5, y=c\_y \* 4)  
 t\_5.place(x=c\_x \* 5, y=c\_y \* 5)  
  
 f\_1 = tk.Label(f\_head, text=fathers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 f\_2 = tk.Label(f\_head, text=fathers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 f\_3 = tk.Label(f\_head, text=fathers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 f\_4 = tk.Label(f\_head, text=fathers[3], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 f\_5 = tk.Label(f\_head, text=fathers[4], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
  
 f\_1.place(x=c\_x \* 8, y=c\_y)  
 f\_2.place(x=c\_x \* 8, y=c\_y \* 2)  
 f\_3.place(x=c\_x \* 8, y=c\_y \* 3)  
 f\_4.place(x=c\_x \* 8, y=c\_y \* 4)  
 f\_5.place(x=c\_x \* 8, y=c\_y \* 5)  
  
 if l == 2:  
 ans1 = tk.Label(f\_head, text=answers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans1.place(x=c\_x, y=c\_y)  
 e\_ans1 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans1.place(x=c\_x \* 11, y=c\_y)  
  
 ans2 = tk.Label(f\_head, text=answers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans2.place(x=c\_x, y=c\_y \* 2)  
 e\_ans2 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans2.place(x=c\_x \* 11, y=c\_y \* 2)  
 if l == 3:  
 ans1 = tk.Label(f\_head, text=answers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans1.place(x=c\_x, y=c\_y)  
 e\_ans1 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans1.place(x=c\_x \* 11, y=c\_y)  
  
 ans2 = tk.Label(f\_head, text=answers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans2.place(x=c\_x, y=c\_y \* 2)  
 e\_ans2 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans2.place(x=c\_x \* 11, y=c\_y \* 2)  
  
 ans3 = tk.Label(f\_head, text=answers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans3.place(x=c\_x, y=c\_y \* 3)  
 e\_ans3 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans3.place(x=c\_x \* 11, y=c\_y \* 3)  
  
 if l == 4:  
 ans1 = tk.Label(f\_head, text=answers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans1.place(x=c\_x, y=c\_y)  
 e\_ans1 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans1.place(x=c\_x \* 11, y=c\_y)  
  
 ans2 = tk.Label(f\_head, text=answers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans2.place(x=c\_x, y=c\_y \* 2)  
 e\_ans2 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans2.place(x=c\_x \* 11, y=c\_y \* 2)  
  
 ans3 = tk.Label(f\_head, text=answers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans3.place(x=c\_x, y=c\_y \* 3)  
 e\_ans3 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans3.place(x=c\_x \* 11, y=c\_y \* 3)  
  
 ans4 = tk.Label(f\_head, text=answers[3], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans4.place(x=c\_x, y=c\_y \* 4)  
 e\_ans4 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans4.place(x=c\_x \* 11, y=c\_y \* 4)  
 if l == 5:  
 ans1 = tk.Label(f\_head, text=answers[0], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans1.place(x=c\_x, y=c\_y)  
 e\_ans1 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans1.place(x=c\_x \* 11, y=c\_y)  
  
 ans2 = tk.Label(f\_head, text=answers[1], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans2.place(x=c\_x, y=c\_y \* 2)  
 e\_ans2 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans2.place(x=c\_x \* 11, y=c\_y \* 2)  
  
 ans3 = tk.Label(f\_head, text=answers[2], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans3.place(x=c\_x, y=c\_y \* 3)  
 e\_ans3 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans3.place(x=c\_x \* 11, y=c\_y \* 3)  
  
 ans4 = tk.Label(f\_head, text=answers[3], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans4.place(x=c\_x, y=c\_y \* 4)  
 e\_ans4 = tk.Entry(f\_head, width=40, bd=5)  
 e\_ans4.place(x=c\_x \* 11, y=c\_y \* 4)  
  
 ans5 = tk.Label(f\_head, text=answers[4], fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 ans5.place(x=c\_x, y=c\_y \* 5)  
 e\_ans5 = tk.Entry(f\_head, width=40,bd=5)  
 e\_ans5.place(x=c\_x \* 11, y=c\_y \* 5)  
  
 def insert\_all\_feed():  
 if l == 2:  
 if e\_ans1.get() == '' or e\_ans2.get() == '':  
 messagebox.showinfo("Error", "You must fill up both fields with their respective feedback.", parent=f\_head)  
 return  
 add\_feed\_to\_DB(quest\_id, answers[0], e\_ans1.get(), m\_id)  
 add\_feed\_to\_DB(quest\_id, answers[1], e\_ans2.get(), m\_id)  
 if l == 3:  
 if e\_ans1.get() == '' or e\_ans2.get() == '' or e\_ans3.get():  
 messagebox.showinfo("Error", "You must fill up both fields with their respective feedback.", parent=f\_head)  
 return  
 add\_feed\_to\_DB(quest\_id, answers[0], e\_ans1.get(), m\_id)  
 add\_feed\_to\_DB(quest\_id, answers[1], e\_ans2.get(), m\_id)  
 add\_feed\_to\_DB(quest\_id, answers[2], e\_ans3.get(), m\_id)  
 if l == 4:  
 if e\_ans1.get() == '' or e\_ans2.get() == '' or e\_ans3.get() or e\_ans4.get():  
 messagebox.showinfo("Error", "You must fill up both fields with their respective feedback.", parent=f\_head)  
 return  
 add\_feed\_to\_DB(quest\_id, answers[0], e\_ans1.get(), m\_id)  
 add\_feed\_to\_DB(quest\_id, answers[1], e\_ans2.get(), m\_id)  
 add\_feed\_to\_DB(quest\_id, answers[2], e\_ans3.get(), m\_id)  
 add\_feed\_to\_DB(quest\_id, answers[3], e\_ans4.get(), m\_id)  
 if l == 5:  
 if e\_ans1.get() == '' or e\_ans2.get() == '' or e\_ans3.get() or e\_ans4.get() or e\_ans5.get():  
 messagebox.showinfo("Error", "You must fill up both fields with their respective feedback.", parent=f\_head)  
 return  
 add\_feed\_to\_DB(quest\_id, answers[0], e\_ans1.get(), m\_id)  
 add\_feed\_to\_DB(quest\_id, answers[1], e\_ans2.get(), m\_id)  
 add\_feed\_to\_DB(quest\_id, answers[2], e\_ans3.get(), m\_id)  
 add\_feed\_to\_DB(quest\_id, answers[3], e\_ans4.get(), m\_id)  
 add\_feed\_to\_DB(quest\_id, answers[4], e\_ans5.get(), m\_id)  
  
 feed\_form.destroy()  
  
 feed\_form\_sub = tk.Button(f\_head, text="Submit", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), command=insert\_all\_feed)  
 feed\_form\_sub.place(x=500, y=500)  
  
 def register\_mod\_DB(mod\_name, start\_quest, ans, inc\_ans, currForm, typeOfQuestion, mark, mod\_exist=False):  
 #  
 # 1. We add the module to database so it has a Key  
 # 2. We find that key  
 # 3. We can now add questions to its own table having access to the specific foreign key (mod\_id)  
 #  
 # Add otherAnswer and times attribute when adding to DB  
  
 if typeOfQuestion == 'tf':  
 # ADD MODULE TO DB  
 onlyDeleteBtnModules()  
 if mod\_exist is False:  
 add\_mod(mod\_name)  
  
 e6 = findModId(mod\_name)  
 # ADD QUESTIONS TO DB  
 add\_quest(start\_quest, e6, inc\_ans, ans, mark, typeOfQuestion)  
  
 q\_id = find\_quest\_id(start\_quest)  
 delBtnModsAndUpdate()  
 # ADD FEEDBACK TO DB  
 all\_ans = [ans, inc\_ans]  
 # hacky way of inserting question id to the feedback  
 add\_feed\_frame(q\_id, all\_ans, currForm, e6)  
 elif typeOfQuestion == 'mcq':  
 all\_ans = inc\_ans.split(',') + ans.split(',')  
 onlyDeleteBtnModules()  
 if mod\_exist is False:  
 add\_mod(mod\_name)  
 e6 = findModId(mod\_name)  
 add\_quest(start\_quest, e6, inc\_ans, ans, mark, typeOfQuestion)  
 q\_id = find\_quest\_id(start\_quest)  
 delBtnModsAndUpdate()  
 add\_feed\_frame(q\_id, all\_ans, currForm, e6)  
 elif typeOfQuestion == 'bm':  
 # NEED TO CONSIDER TWO THINGS  
 # The Admin can add some options  
 # Each of these options can have some answers  
 # the app should store these answers and options  
 # How can we link these options and answer to determine which belongs to which?  
 # IDEA1: Create a new Answers Table that will hold the answers and its option.  
 # This way we can just fetch this entities and compare them with what the user matched in the app.  
 # Idea 2: ADD an "A" To the beginning of each option and Answer so we know they are connected.  
 # We would just need to check the first letter to determine the correctness of the user match.  
 onlyDeleteBtnModules()  
 if mod\_exist is False:  
 add\_mod(mod\_name)  
 e6 = findModId(mod\_name)  
 add\_quest(start\_quest, e6, inc\_ans, ans, mark, typeOfQuestion)  
 delBtnModsAndUpdate()  
 get\_inc\_ans\_from\_bma\_frame(start\_quest, typeOfQuestion, ans, currForm)  
  
 add\_mod\_btn = tk.Button(head, text="New Module",fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), command=chooseTypeOfQuestion)  
 add\_mod\_btn.grid(row=0, column=1, padx=(100, 0))  
  
 # -----------------ORDER TO DELETE A MODULE----------------  
 # 1. Delete All questions from Databases  
 # 2. Delete Module From database  
 # 3. Close window frame and DeleteAndUpdate  
 #  
 def delModuleFrame():  
 are\_there\_modules = fetchModules()  
 if len(are\_there\_modules) == 0:  
 messagebox.showinfo("Error", "There are not modules available to delete." + "\n" + "Add modules first.")  
 return  
 window = tk.Tk()  
 window.configure(bg='#5D9DE5')  
 window.resizable(0, 0)  
 window.geometry("500x200")  
 window.title("Delete Module")  
 currModule = tk.StringVar(window)  
 currModule.set("--Select Module--") # default value  
  
 # Here the dropdown menu is created  
 # using the 'data' modules  
 # from the database.  
 chooseTest = tk.OptionMenu(window, currModule, \*fetchModules())  
 chooseTest.config(fg="white", bg='#5D9DE5', font=('Helvetica', 11, 'bold'))  
 menu = window.nametowidget(chooseTest.menuname)  
 menu.config(fg="white", bg='#5D9DE5', font=('Helvetica', 11, 'bold'))  
 chooseTest.place(x=20, y=20)  
  
 def deleteModule():  
 # ----------------MAIN DELETE MODULE FUNCTION--------------  
  
 # In order to make the app more dynamic.  
 # 1. Remove all widget buttons from the frame  
 # 2. Delete all feedback from the db of feedback from module chosen id  
 # 2. Delete all questions from the db of questions from module chosen  
 # 3. Delete Module from DB  
 # 4. Display all buttons widgets again  
 # 5. Delete frame window itself  
 onlyDeleteBtnModules()  
 mod\_to\_delete = toText(currModule.get())  
 get\_mod\_id = findModId(mod\_to\_delete)  
 delAllFeedbackFromDB(get\_mod\_id)  
 delAllBmaFromDB(get\_mod\_id)  
 delAllQuestionsFromDB(get\_mod\_id)  
 delModFromDB(mod\_to\_delete)  
 delBtnModsAndUpdate()  
 window.destroy()  
  
 sub\_del = tk.Button(window, text="erase", command=deleteModule, fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 sub\_del.place(x=230, y=20)  
  
 alert\_txt = tk.Label(window, text="Careful! All questions & feedback will also be erased",  
 fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'))  
 alert\_txt.place(x=20, y=120)  
  
 del\_mod\_btn = tk.Button(head, text="Del Module", fg="white", bg='#5D9DE5', font=('Helvetica', 13, 'bold'), command=delModuleFrame)  
 del\_mod\_btn.grid(row=0, column=2, padx=(10, 0))  
  
  
class QuizzApp(tk.Tk):  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 tk.Tk.\_\_init\_\_(self, \*args, \*\*kwargs)  
  
 self.currScore = 0  
 # Create the window  
 window = tk.Frame(self)  
 window.pack()  
  
 window.grid\_rowconfigure(0, minsize=700)  
 window.grid\_columnconfigure(0, minsize=700)  
  
 """  
 This idea of maintaining the frames in a constant for loop came from this video.  
 https://www.youtube.com/watch?v=tpGjHRDEjCE&t=1153s&ab\_channel=IGTechTeam  
   
 Basically, it creates a dictionary to store all the classes of the questionnaire.  
 Then with the "change\_frame" function it changes to which class you want to visit.   
 """  
 self.containerOfFrames = {}  
 for f in (LogInPage, UserHomePage, AdminHomePage):  
 frame = f(window, self)  
 self.containerOfFrames[f] = frame  
 frame.grid(row=0, column=0, sticky="nsew")  
 self.change\_frame(LogInPage)  
 def exitSystem(self):  
 self.destroy()  
 # self.containerOfFrames[LogInPage].destroy()  
 # self.containerOfFrames[AdminHomePage].destroy()  
 # self.containerOfFrames[UserHomePage].destroy()  
 def change\_frame(self, page):  
 frame = self.containerOfFrames[page]  
 frame.tkraise()  
  
 """  
 This function can be used to get the score of the user at any point in time whenever a  
 quiz has been initialized.  
 There is gotta be another function to update the score to 0.  
   
 """  
  
 def getScore(self):  
 print(self.currScore)  
  
  
app = QuizzApp()  
app.mainloop()