

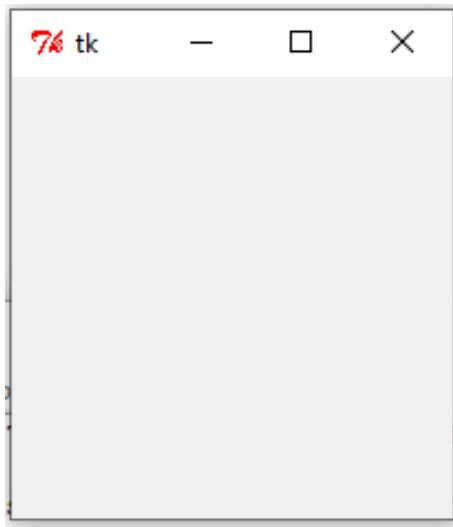
# Development

## Main Menu

### Main Menu GUI

From my design, I should start developing a solution by making a main menu first. **Otherwise, I will not be able to start developing the other parts of the program**

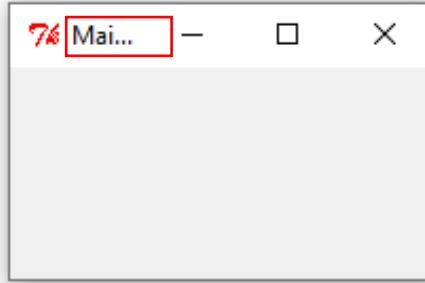
I will use 'Tkinter' to create a GUI in python. **Otherwise, the program cannot be implemented into a graphical form.**



```
from tkinter import *
root=Tk()
root.mainloop()
```

Next, I add a suitable title and resize the window. **Otherwise, the purpose of the window is not clear and without resizing the window the title is not fully visible.**

```
from tkinter import *
root=Tk()
root.geometry("200x100")
root.title("Main Menu")
root.mainloop()
```

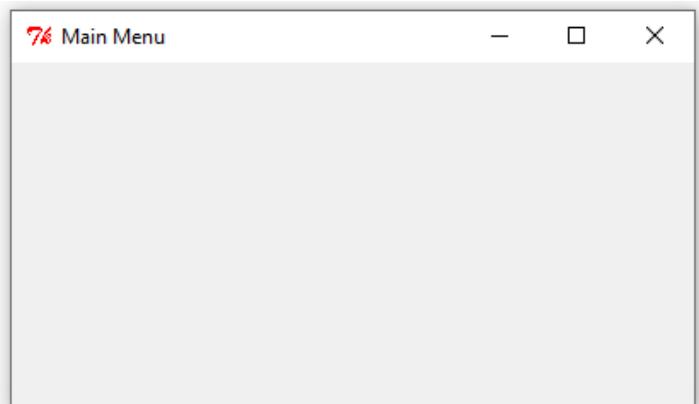


Window is resized as  
the title does not fit  
on the screen.  
**Otherwise, the user  
cannot see the title**

I make the window larger for the title of the window to fit. **Otherwise, it will not be easy to see the window is being used**

```
from tkinter import *
root=Tk()
root.geometry("400x200")
root.title("Main Menu")
root.mainloop()
```

I make the window larger for the title of the window to fit.  
Otherwise it will not



## Error

I try to add a title within the window (**otherwise, the purpose of the window is not fully clear**), but I mixed up the variable name and command

```
from tkinter import *

root=Tk()
root.geometry("400x200")
root.title("Main Menu")

Label=label(root,text="Welcome to the main menu")
label.pack()

root.mainloop()
```

'label' should be 'Label'

```
from tkinter import *

root=Tk()
root.geometry("400x200")
root.title("Main Menu")

Label=label(root,text="Welcome to the main menu")
label.pack()

root.mainloop()
```

Label=label() should turn into  
label=Label()

As 'label' is the variable name  
and 'Label' is the widget

**74 Python Shell**

---

File Edit Shell Debug Options Windows Help

Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v. 32 bit (Intel)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ===== RESTART =====

>>>

Traceback (most recent call last):

File "N:\GUI.py", line 7, in <module>

    Label=label(root,text="Welcome to the main menu")

NameError: name 'label' is not defined

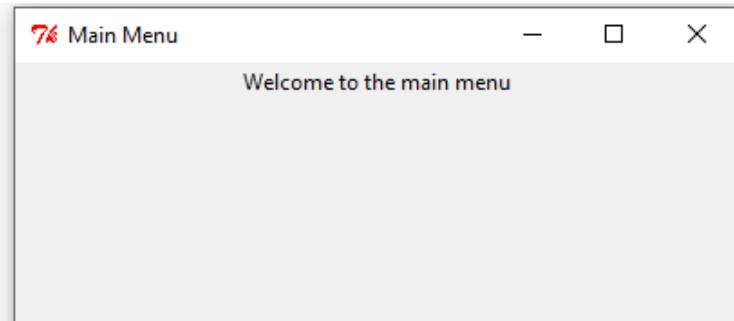
>>>

The label is added correctly. **Otherwise, an error will appear and without the label, the purpose of the program is not fully clear.**

```
from tkinter import *
root=Tk()
root.geometry("400x200")
root.title("Main Menu")

label1=Label(root,text="Welcome to the main menu")
label1.pack()

root.mainloop()
```



I add one button to register and one button to login. **Otherwise, the user will not be able to select an option using the GUI.**

```
from tkinter import *

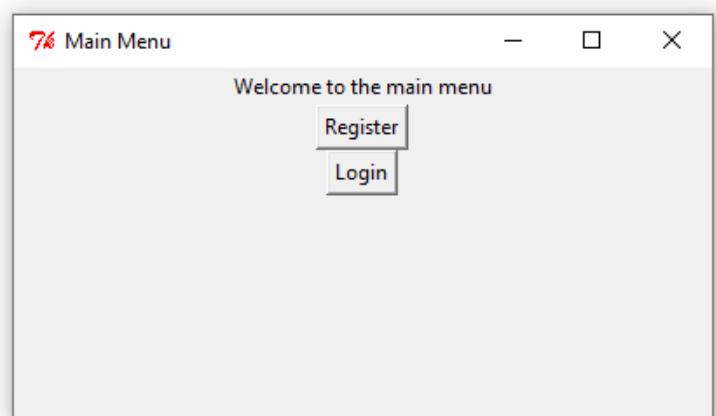
root=Tk()
root.geometry("400x200")
root.title("Main Menu")

label1=Label(root,text="Welcome to the main menu")
label1.pack()

buttonRegister=Button(root,text="Register")
buttonRegister.pack()

buttonLogin=Button(root,text="Login")
buttonLogin.pack()

root.mainloop()
```



I separate the buttons to prevent accidentally pressing the wrong one. **Otherwise, it will be easy for a user to accidentally press the wrong button**

```
from tkinter import *

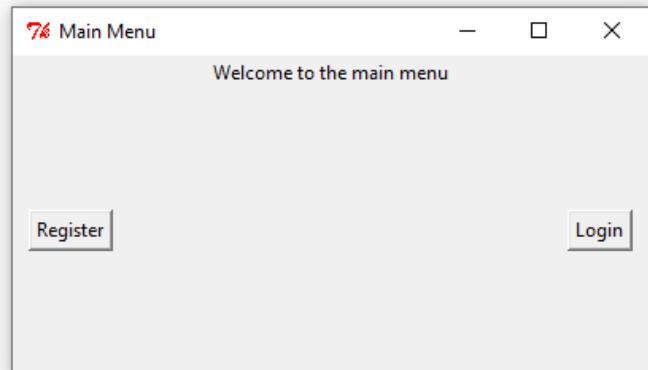
root=Tk()
root.geometry("400x200")
root.title("Main Menu")

label1=Label(root,text="Welcome to the main menu")
label1.pack()

buttonRegister=Button(root,text="Register")
buttonRegister.pack(side=LEFT,padx=10,pady=10)

buttonLogin=Button(root,text="Login")
buttonLogin.pack(side=RIGHT,padx=10,pady=10)

root.mainloop()
```



## Error

I try to change the background colour (**otherwise, the design of the window looks dull**) using a hex value, but forget to start with a hashtag

```
from tkinter import *
root=Tk()
root.geometry("400x200")
root.title("Main Menu")
root.configure(bg="#33BFFF")

label1=Label(root,text="Welcome to the main menu")
label1.pack(side=TOP)

buttonRegister=Button(root,text="Register")
buttonRegister.pack(side=LEFT,padx=10,pady=10)

buttonLogin=Button(root,text="Login")
buttonLogin.pack(side=RIGHT,padx=10,pady=10)

root.mainloop()
```

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] (32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Traceback (most recent call last):
  File "N:\GUI.py", line 6, in <module>
    root.configure(bg="33BFFF")
File "C:\Python32\lib\tkinter\_init__.py", line 1193, in configure
  return self._configure('configure', cnf, kw)
File "C:\Python32\lib\tkinter\_init__.py", line 1184, in _configure
  self.tk.call(_flatten((self._w, cmd)) + self._options(cnf))
_tkinter.TclError: unknown color name "33BFFF"
>>> |
```

The background colour is changed correctly. **Otherwise, an error will appear and without the background colour, the design of the window looks dull**

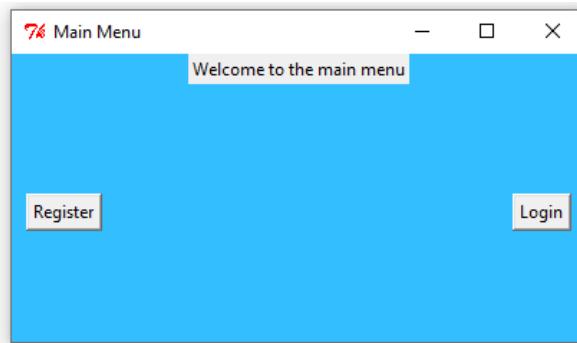
```
from tkinter import *
root=Tk()
root.geometry("400x200")
root.title("Main Menu")
root.configure(bg="#33BFFF")

label1=Label(root,text="Welcome to the main menu")
label1.pack(side=TOP)

buttonRegister=Button(root,text="Register")
buttonRegister.pack(side=LEFT,padx=10,pady=10)

buttonLogin=Button(root,text="Login")
buttonLogin.pack(side=RIGHT,padx=10,pady=10)

root.mainloop()
```



# added to correct the format of the colour

I replace the .pack() command with .place() to be able to place my buttons more precisely (**otherwise, I will find it difficult to place buttons**). However, the 'Login' button is not in completely visible.

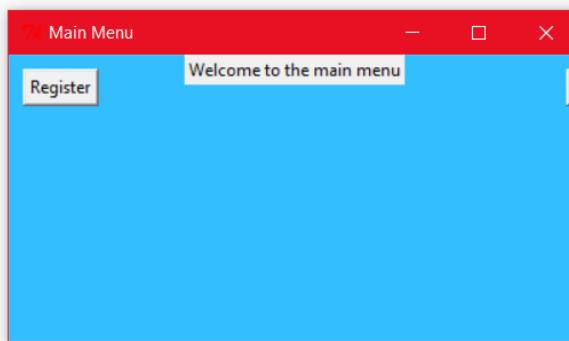
```
from tkinter import *
root=Tk()
root.geometry("400x200")
root.title("Main Menu")
root.configure(bg="#33BFFF")

label1=Label(root,text="Welcome to the main menu")
label1.pack(side=TOP)

buttonRegister=Button(root,text="Register")
buttonRegister.place(x=10, y=10)

buttonLogin=Button(root,text="Login")
buttonLogin.place(x=390,y=10)

root.mainloop()
```



Window resized to fit 'Login' button

I correctly position the buttons. **Otherwise, the login button is not fully visible by the user**

```
from tkinter import *
root=Tk()
root.geometry("400x200")
root.title("Main Menu")
root.configure(bg="#33BFFF")

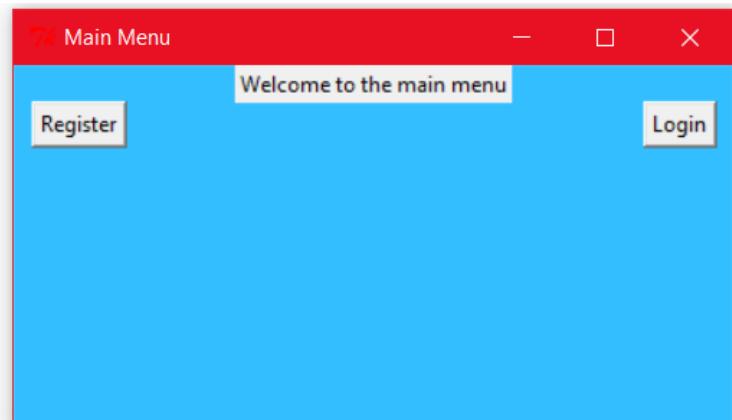
label1=Label(root,text="Welcome to the main menu")
label1.pack(side=TOP)

buttonRegister=Button(root,text="Register")
buttonRegister.place(x=10, y=20)

buttonLogin=Button(root,text="Login")
buttonLogin.place(x=350,y=20)

root.mainloop()
```

x value is changed so the button fully fits on screen



I clarify the purpose of both buttons, by changing the text. I also resize the window and move the login button. **Otherwise, the purpose of the buttons is harder to understand**

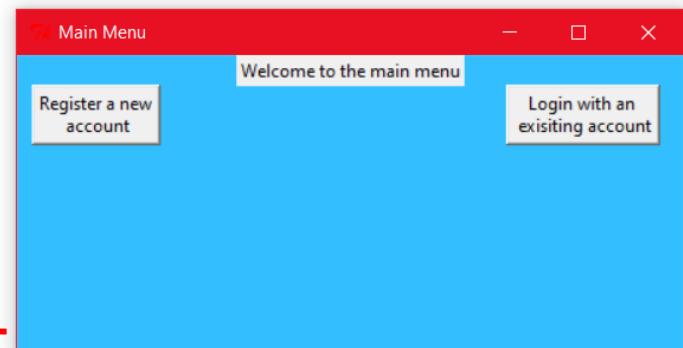
```
from tkinter import *
root=Tk()
root.geometry("450x200")
root.title("Main Menu")
root.configure(bg="#33BFFF")

label1=Label(root,text="Welcome to the main menu")
label1.pack(side=TOP)

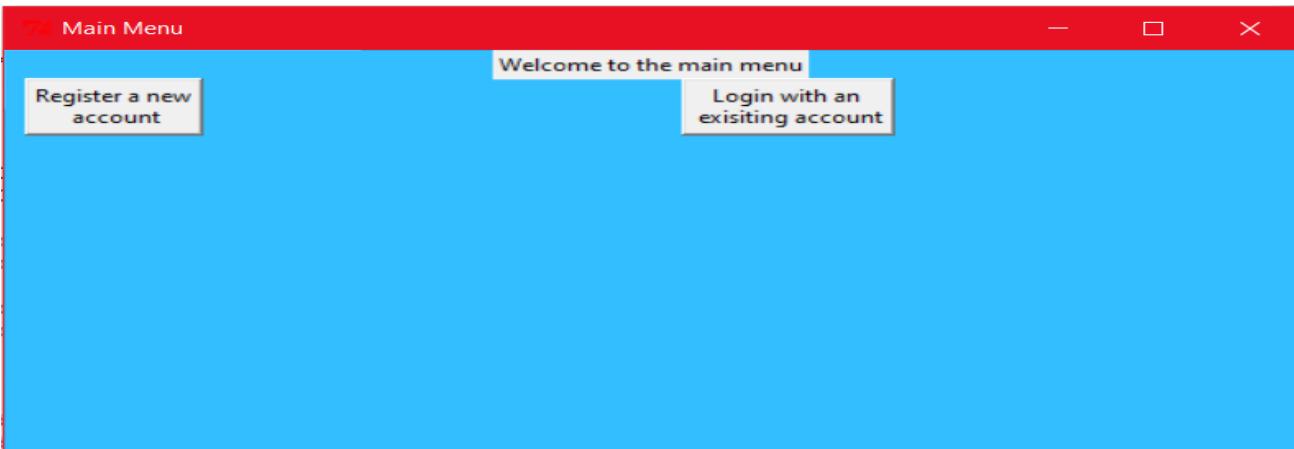
buttonRegister=Button(root,text="Register a new\naccount")
buttonRegister.place(x=10, y=20)

buttonLogin=Button(root,text="Login with an\nexisting account")
buttonLogin.place(x=330,y=20)

root.mainloop()
```



I realise the buttons do not move relative to the size of the window when resizing it by dragging the edges of the window. **Without this problem, the buttons will always be aligned correctly**



The label is displayed using the `.pack()` command, this always centers the label even if the window is resized. However since the buttons are displayed using the `.place()` command, they remain in their given x and y coordinate positions as it is not affected by the window's size.

To fix this, I disable the ability to resize the window manually. This prevents the window from being resized by the user, making sure that all buttons are always visible and aligned correctly. **Otherwise, the user could resize the window, which would hide features of the window**

```
from tkinter import *
root=Tk()
root.geometry("450x200")
root.title("Main Menu")
root.configure(bg="#33BFFF")

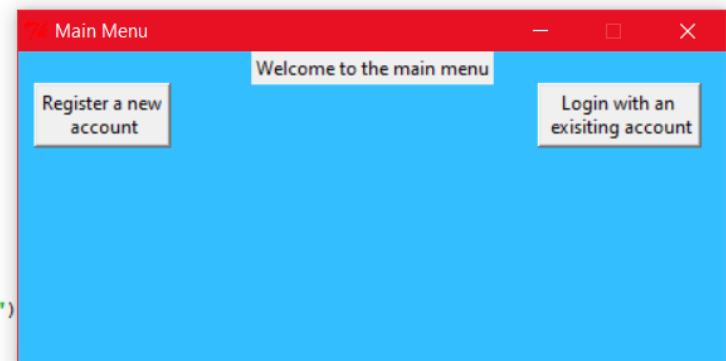
root.resizable(False,False)

label=Label(root,text="Welcome to the main menu")
label.pack(side=TOP)

buttonRegister=Button(root,text="Register a new\n account")
buttonRegister.place(x=10, y=20)

buttonLogin=Button(root,text="Login with an\n existing account")
buttonLogin.place(x=330,y=20)

root.mainloop()
```



I add a 'Guide' and 'Exit' Button . **Otherwise, the user would not be able to access the guide or exit the program easily.**

```

from tkinter import *

root=Tk()
root.geometry("450x200")
root.title("Main Menu")
root.configure(bg="#33BFFF")

root.resizable(False, False)

labell=Label(root, text="Welcome to the main menu")
labell.pack(side=TOP)

buttonRegister=Button(root, text="Register a new\n account")
buttonRegister.place(x=10, y=20)

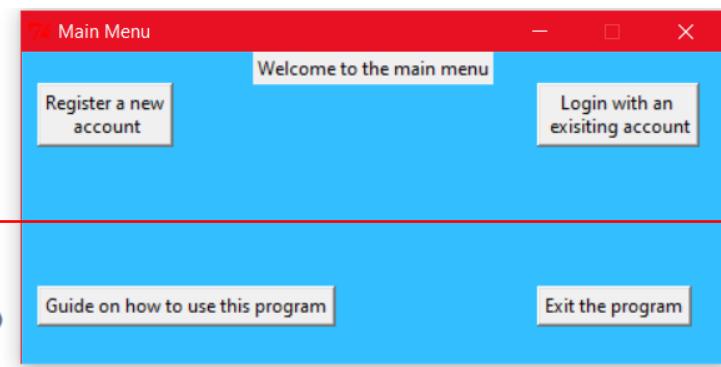
buttonLogin=Button(root, text="Login with an\n existing account")
buttonLogin.place(x=330, y=20)

buttonEnd=Button(root, text="Exit the program")
buttonEnd.place(x=330, y=150)

buttonGuide=Button(root, text="Guide on how to use this program")
buttonGuide.place(x=10, y=150)

root.mainloop()

```



## Error

I try to make the 'Guide' button more discrete (**without this all buttons will look the same, making the design dull**), like as shown in my design. However, I get an error message.

```

buttonGuide=Button(root, text="Guide on how to use this program", bg="#33BFFF", font=("Courier underline"))
buttonGuide.place(x=10, y=150)

```

The 'font' command has 2 necessary arguments: the font name and size. Therefore, the command always expects the second argument to be an integer, when given a property such as 'underline', an error is given.

```

Traceback (most recent call last):
  File "C:\Users\acer\Desktop\Computing\Computing\GUI.py", line 22, in <module>
    buttonGuide=Button(root, text="Guide on how to use this program", bg="#33BFFF"
", font=("Courier underline"))
  File "C:\Python32\lib\tkinter\__init__.py", line 2028, in __init__
    Widget.__init__(self, master, 'button', cnf, kw)
  File "C:\Python32\lib\tkinter\__init__.py", line 1958, in __init__
    (widgetName, self._w) + extra + self._options(cnf))
_tkinter.TclError: expected integer but got "underline"

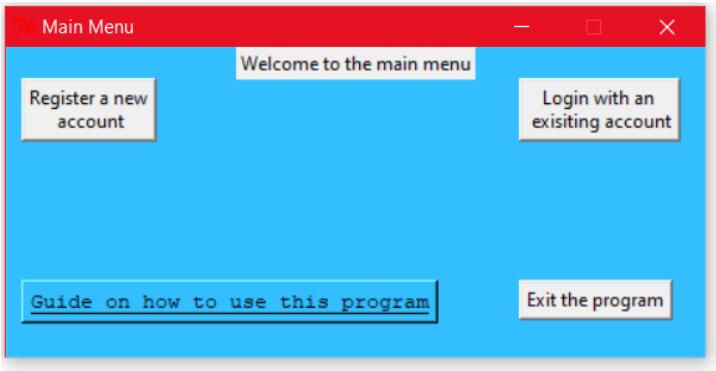
```

I correct the error, by specifying the size of the font. **Otherwise, I would receive an error.**

```

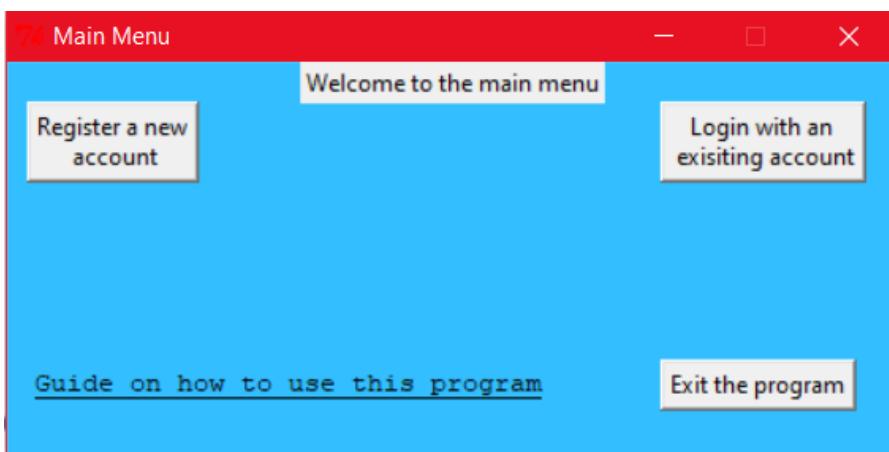
buttonGuide=Button(root, text="Guide on how to use this program", bg="#33BFFF", font=("Courier 10 underline"))
buttonGuide.place(x=10, y=150)

```



I remove the border on the button by reducing the border width to 0. This makes the button blend better with the background. **Otherwise, the button does not look unique.**

```
buttonGuide=Button(root,text="Guide on how to use this program", bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
```



I define the functions connected to the buttons and link these to execute when the button is clicked. **Otherwise, the buttons would not execute their respective functions.**

```
buttonRegister=Button(root,text="Register a new\n account",command=registerFunction)
buttonRegister.place(x=10, y=20)

buttonLogin=Button(root,text="Login with an\n exisiting account",command=loginFunction)
buttonLogin.place(x=330,y=20)

buttonEnd=Button(root,text="Exit the program",command=exitFunction)
buttonEnd.place(x=330,y=150)

buttonGuide=Button(root,text="Guide on how to use this program",command=guideFunction,bg="red")
buttonGuide.place(x=10,y=150)

root.mainloop()

def registerFunction:

def loginFunction:

def exitFunction:

def guideFunction:
```

## Error

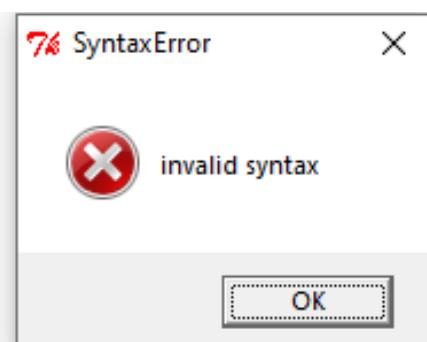
I forgot to add parameters to my functions (**without this an error appears**), which gives me a syntax error.

```
def registerFunction:

def loginFunction:

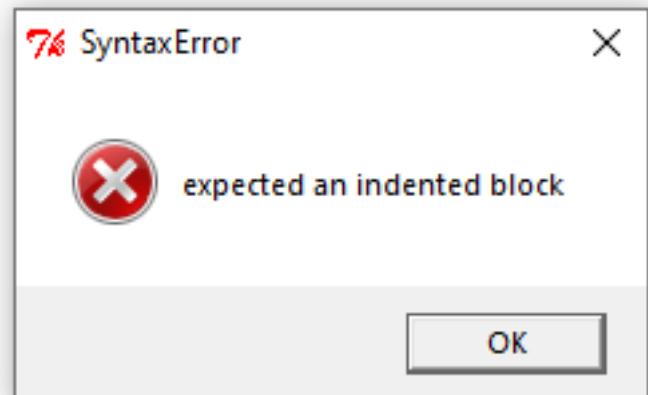
def exitFunction:

def guideFunction:
```



I add parameters to my functions, but get another syntax error

```
def registerFunction():  
  
def loginFunction():  
  
def exitFunction():  
  
def guideFunction():
```



I fix this by disabling the other functions for now by turning the region into a comment. **Otherwise, I would receive a syntax error**

```
def registerFunction():  
  
##def loginFunction():  
##  
##def exitFunction():  
##  
##def guideFunction():  
##
```

This is the end of my first prototype for my main menu. Next, I will create a separate window for registering an account and this will be linked to the 'register' button on the main menu

## Registration

### Registration GUI

#### Error

I try to create a new window for registering an account (**otherwise, the user will not be able to register an account using the GUI**), but I get an error

```
from tkinter import *

root=Tk()
root.geometry("450x200")
root.title("Main Menu")
root.configure(bg="#33BFFF")

root.resizable(False, False)

labell=Label(root, text="Welcome to the main menu")
labell.pack(side=TOP)

buttonRegister=Button(root, text="Register a new\n account", command=registerFunction)
buttonRegister.place(x=10, y=20)

buttonLogin=Button(root, text="Login with an\n existing account", command=loginFunction)
buttonLogin.place(x=330, y=20)

buttonEnd=Button(root, text="Exit the program", command=exitFunction)
buttonEnd.place(x=330, y=150)

buttonGuide=Button(root, text="Guide on how to use this program", command=guideFunction)
buttonGuide.place(x=10, y=150)

root.mainloop()

def registerFunction():
    registerWindow=Tk()
    registerWindow.geometry("450x200")
    registerWindow.title("Registration")
    registerWindow.configure(bg="#33BFFF")|
```

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Traceback (most recent call last):
  File "N:\GUI (1).py", line 13, in <module>
    buttonRegister=Button(root, text="Register a new\n account", command=registerFunction)
NameError: name 'registerFunction' is not defined
>>>
```

## Error

I move the function to before it is assigned to the button (**otherwise, I will receive an error**). This fixes the earlier error but creates a new one as the program reads the button commands, which have not been defined yet, causing an error.

```

def registerFunction():
    registerWindow=Tk()
    registerWindow.geometry("450x200")
    registerWindow.title("Registration")
    registerWindow.configure(bg="#33BFFF")

buttonRegister=Button(root,text="Register a new\n account",command=registerFunction)
buttonRegister.place(x=10, y=20)

buttonLogin=Button(root,text="Login with an\n exisiting account",command=loginFunction)
buttonLogin.place(x=330,y=20)

buttonEnd=Button(root,text="Exit the program",command=exitFunction)
buttonEnd.place(x=330,y=150)

buttonGuide=Button(root,text="Guide on how to use this program",command=guideFunction)
buttonGuide.place(x=10,y=150)

Traceback (most recent call last):
  File "N:\GUI (1).py", line 22, in <module>
    buttonLogin=Button(root,text="Login with an\n exisiting account",command=log
inFunction)
NameError: name 'loginFunction' is not defined
>>>

```

Undefined commands must be removed to remove the error

I fix this error by removing all the empty functions and the commands to the buttons. **Otherwise, I would receive a NameError.** I will add these again later when I add more functions.

```

def registerFunction():
    registerWindow=Tk()
    registerWindow.geometry("450x200")
    registerWindow.title("Registration")
    registerWindow.configure(bg="#33BFFF")

buttonRegister=Button(root,text="Register a new\n account",command=registerFunction)
buttonRegister.place(x=10, y=20)

buttonLogin=Button(root,text="Login with an\n exisiting account")  
buttonLogin.place(x=330,y=20)

buttonEnd=Button(root,text="Exit the program"  )
buttonEnd.place(x=330,y=150)

buttonGuide=Button(root,text="Guide on how to use this program",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)  
buttonGuide.place(x=10,y=150)

root.mainloop()

```

Unused commands removed to prevent error



## Error

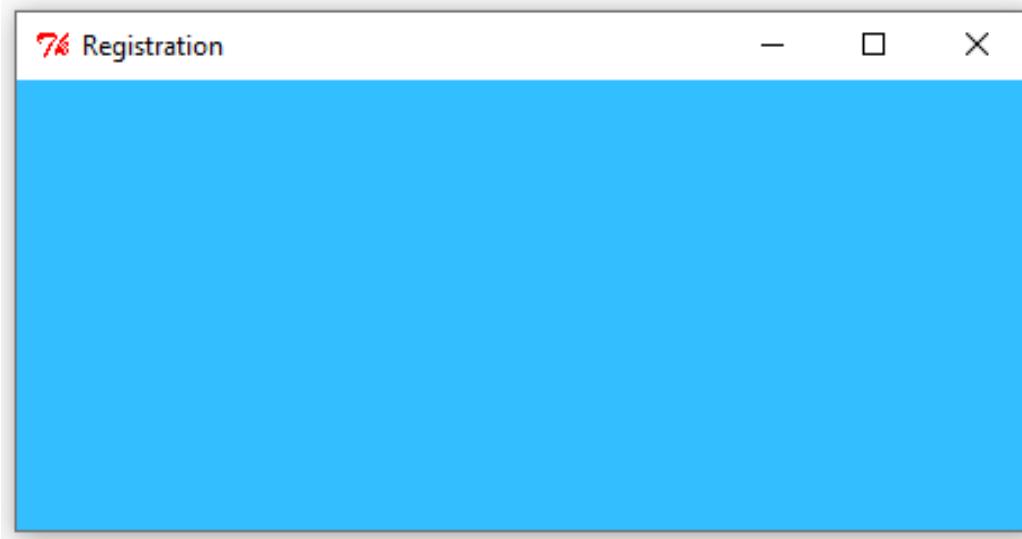
I try to add a label and try to prevent resizing the window (**otherwise the purpose of the window would not be clear and the buttons will not be aligned correctly**), but I get an error.

```
def registerFunction():
    registerWindow=Tk()
    registerWindow.geometry("450x200")
    registerWindow.title("Registration")
    registerWindow.configure(bg="#33BFFF")

    labelRegistration(registerWindow, text="Registration")
    labelRegistration.pack(side=TOP)

    registerWindow.resizable(False, False)
```

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 19, in registerFunction
    labelRegistration(registerWindow, text="Registration")
NameError: global name 'labelRegistration' is not defined
```



## Error

I fix the label by adding the correct label command, but I get another error

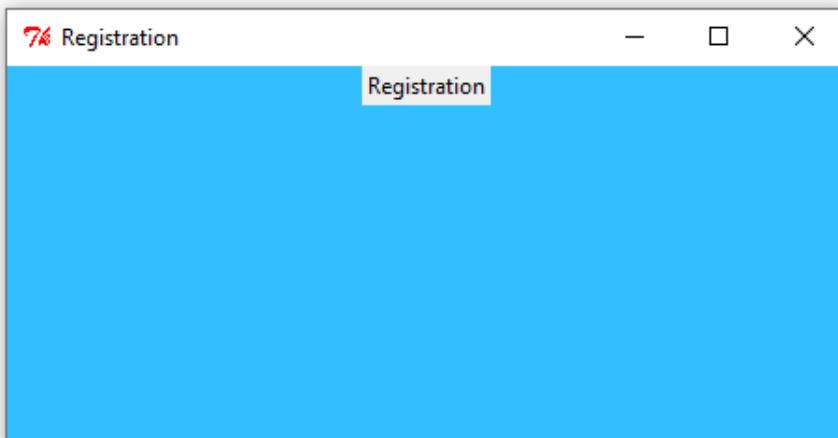
```
labell=Label(root,text="Welcome to the main menu")
labell.pack(side=TOP)

def registerFunction():
    registerWindow=Tk()
    registerWindow.geometry("450x200")
    registerWindow.title("Registration")
    registerWindow.configure(bg="#33BFFF")

    labelRegistration=Label(registerWindow, text="Registration")
    labelRegistration.pack(side=TOP)

    registerWindow.resizable(False, False)
```

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 22, in registerFunction
    registerWindow.resizable(False, False)
  File "C:\Python32\lib\tkinter\__init__.py", line 1750, in __getattr__
    return getattr(self.tk, attr)
AttributeError: 'tkapp' object has no attribute 'resizable'
```



I fix this error by spelling the 'resizable' command correctly. **Without the label at the top of the window, it is hard for a user to tell what the purpose of the window is**

```
label1=Label(root,text="Welcome to the main menu")
label1.pack(side=TOP)

def registerFunction():
    registerWindow=Tk()
    registerWindow.geometry("450x200")
    registerWindow.title("Registration")
    registerWindow.configure(bg="#33BFFF")

    labelRegistration=Label(registerWindow,text="Registration")
    labelRegistration.pack(side=TOP)

    registerWindow.resizable(False, False)
```



## Error

I try to add some input boxes (**otherwise, the user cannot enter their registration details**), but get an error

```
username=tk.entry(registerWindow)
password=tk.entry(registerWindow)

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 24, in registerFunction
    username=tk.entry(registerWindow)
NameError: global name 'tk' is not defined
```

## Error

I try to fix this by calling the Tkinter function under the correct name, but get another error

```
username=Tk.entry(registerWindow)
password=Tk.entry(registerWindow)

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 24, in registerFunction
    username=Tk.entry(registerWindow)
AttributeError: type object 'Tk' has no attribute 'entry'
```

I fix this by correcting the 'entry' command and packing both buttons. I also change the name of the entries as I will use the variables 'username' and 'password' later. **Otherwise, my variable names will be hard to understand and easy to mix up**

```
usernameBox=Entry(registerWindow)
usernameBox.pack()

passwordBox=Entry(registerWindow)
passwordBox.pack()
```



I add some labels to the input boxes. **Otherwise, the user will not know what information to enter into the input boxes**

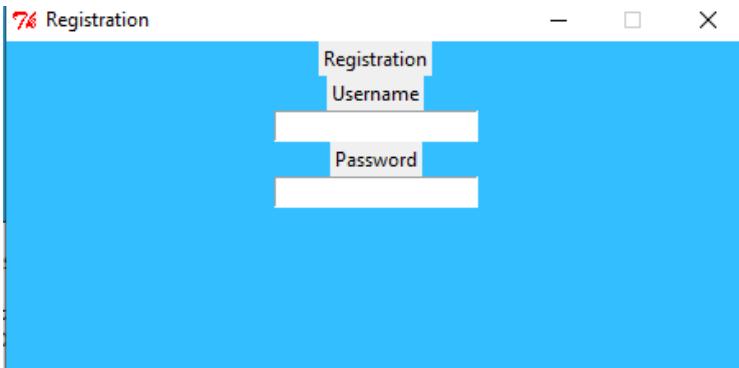
```
labelUsername=Label(registerWindow, text="Username")
labelUsername.pack(side=TOP)

usernameBox=Entry(registerWindow)
usernameBox.pack()

labelPassword=Label(registerWindow, text="Password")
labelPassword.pack(side=TOP)

passwordBox=Entry(registerWindow)
passwordBox.pack()
```

Informative labels added. **Otherwise, the user will not know what to enter**



I separate the labels and input boxes. **Otherwise, the buttons and labels look too cramped.**

```
labelRegistration.pack(side=TOP)

registerWindow.resizable(False, False)

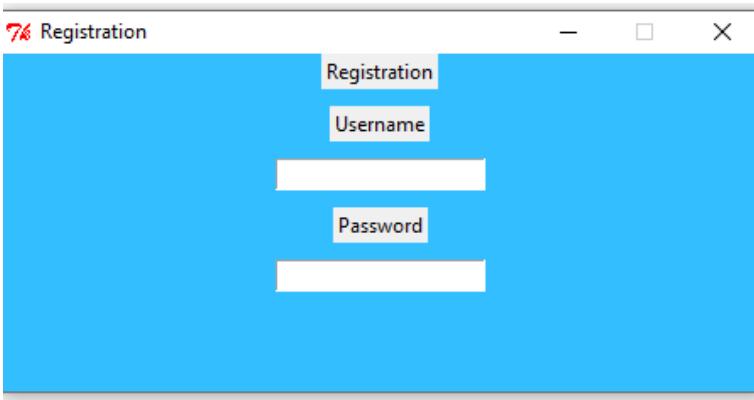
labelUsername=Label(registerWindow, text="Username")
labelUsername.pack(side=TOP,pady=10)

usernameBox=Entry(registerWindow)
usernameBox.pack()

labelPassword=Label(registerWindow, text="Password")
labelPassword.pack(side=TOP,pady=10)

passwordBox=Entry(registerWindow)
passwordBox.pack()
```

GUI elements seperated



I add a 'register' button, **otherwise the user cannot register their account once they have filled in their details.** I also change the 'pack' command to the 'place' command for the 'Username' and 'Password' labels (**without this I will not have as much control over the positioning of the labels**), but I realise I need to do the same for the input boxes and button.

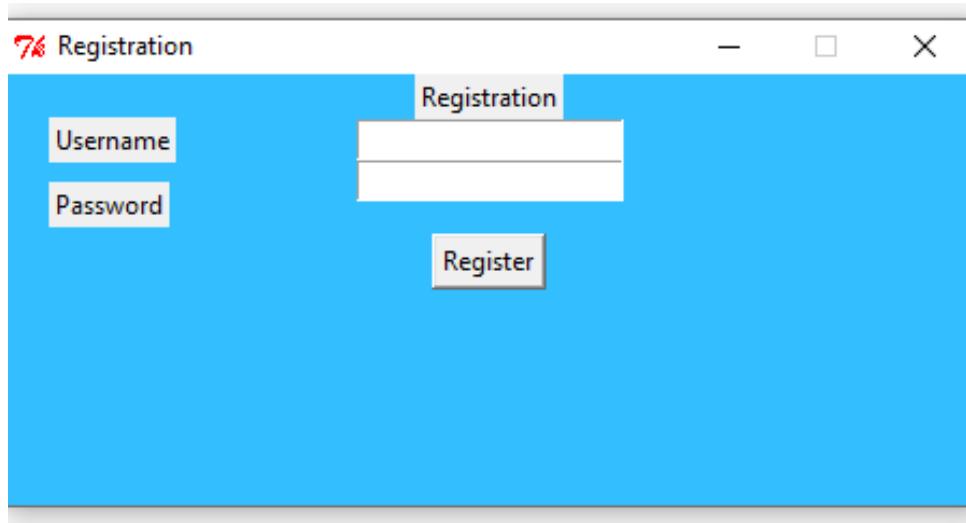
```
labelUsername=Label(registerWindow, text="Username")
labelUsername.place(x=20,y=20)

usernameBox=Entry(registerWindow)
usernameBox.pack()

labelPassword=Label(registerWindow, text="Password")
labelPassword.place(x=20,y=50)

passwordBox=Entry(registerWindow)
passwordBox.pack()

buttonRegister=Button(registerWindow, text="Register")
buttonRegister.pack(pady=15)
```



I rearrange all labels, input boxes and buttons. **Otherwise, the labels are not aligned with their respective buttons, making it hard to understand what to enter in the input boxes.**

```
labelUsername=Label(registerWindow, text="Username")
labelUsername.place(x=20,y=20)

usernameBox=Entry(registerWindow)
usernameBox.place(x=20, y=40)

labelPassword=Label(registerWindow, text="Password")
labelPassword.place(x=20,y=80)

passwordBox=Entry(registerWindow)
passwordBox.place(x=20,y=100)

buttonRegister=Button(registerWindow, text="Register")
buttonRegister.place(x=200,y=150)
```

GUI elements  
seperated



I change the text for the labels, and expand the input boxes. **Otherwise, the GUI looks empty due to the extra space.**

```
labelUsername=Label(registerWindow, text="Username:", bg="#33BFFF", font=("Courier 10 underline"), borderwidth=0)
labelUsername.place(x=20, y=20)

usernameBox=Entry(registerWindow, width=67)
usernameBox.place(x=20, y=40)

labelPassword=Label(registerWindow, text="Password:", bg="#33BFFF", font=("Courier 10 underline"), borderwidth=0)
labelPassword.place(x=20, y=80)

passwordBox=Entry(registerWindow, width=67)
passwordBox.place(x=20, y=100)

buttonRegister=Button(registerWindow, text="Register")
buttonRegister.place(x=200, y=150)
```

GUI elements modified to make it easier to user and look better



I create an excel csv file called 'login'. The file has 3 headings: Username, Password and Admin. This will hold all the login details. **Otherwise, the login details cannot be saved.**

	A	B	C
1	Username	Password	Admin
2			
3			
4			
5			
6			
7			
8			
9			

## Open login file function

I create a new function to open the login file as I will need to access it several times throughout the program. I also import the 'csv' module to be able to access the .csv file. **Otherwise, the program would not be able to access the csv file**

```
def openLogin():
    from tkinter import *
    import csv
```

I make the function open the .csv file and add the contents to a dictionary.

```
def openLogin():
    csvFile=open('login.csv')
    csvFileContent=(csvFile)
    next(csvFile)

    DatabaseList=[]

    for row in csvFileContent:
        database=row.strip().split(",")
        heading=['Username','Password','Admin']
        data=zip(heading,database)
        DataDict=dict(data)
        DatabaseList.append(DataDict)

    login=[]
```

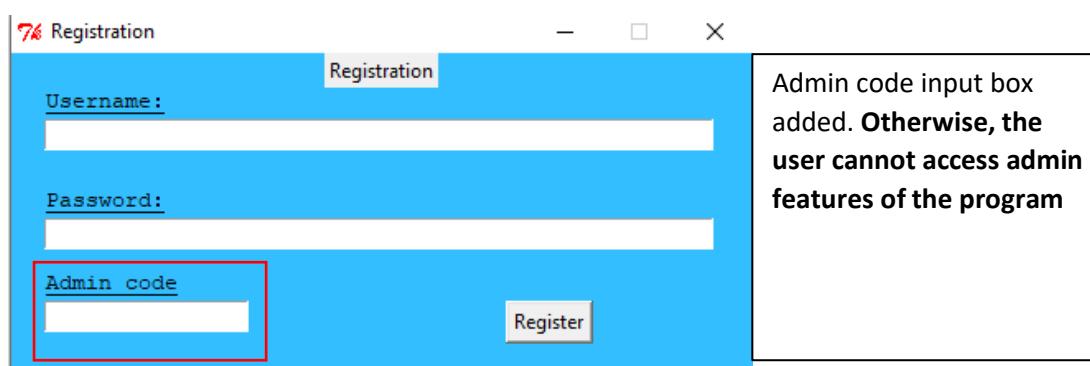
I make this function execute when the 'register' button is selected from the main menu. **Otherwise, the registered details cannot be saved to the csv file**

```
def registerFunction():
    openLogin()
    registerWindow=Tk()
    registerWindow.geometry("450x200")
    registerWindow.title("Registration")
    registerWindow.configure(bg="#33BFFF")
```

I look back at my design and realise I need to add an admin code to decide an account's permissions. **Otherwise, I will not be able to decide whether an account should be able to access the special features of the program.**

```
labelAdmin=Label(registerWindow, text="Admin code",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
labelAdmin.place(x=20,y=130)

adminBox=Entry(registerWindow,width=20)
adminBox.place(x=20,y=150)
```



## Retrieve and save registration details

I create a function to add the inputted details to the .csv file. **Otherwise, the registered details will not be read**

```
def registerAccount():
    username=usernameBox.get()
    password=passwordBox.get()
    admin=adminBox.get()
    logindata={'Username':username,'Password':password,'Admin':admin}
    login.append(logindata)|
```

I create the function to save the login details and make this execute once the login details are submitted. **Otherwise, the registered details will not be saved**

```
def registerAccount():
    username=usernameBox.get()
    password=passwordBox.get()
    admin=adminBox.get()
    logindata={'Username':username,'Password':password,'Admin':admin}
    login.append(logindata)
    saveLoginData()

def saveLoginData():
    try:
        fileHandle=open('login.csv','r+')
        fileContent=fileHandle.read()
        print(fileContent)
        if fileContent.strip()=='':
            fileHandle.write('Username,Password,Admin\n')
        for item in login:
            fileHandle.write('{Username},{Password},{Admin}'.format(**item))
            fileHandle.write('\n')
        fileHandle.close()
    except OSError:
        print('Can\'t write to file')
```

## Error

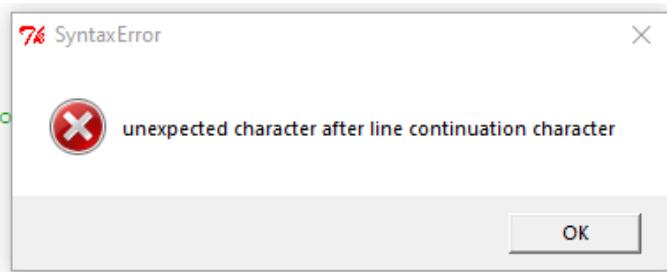
I get a syntax error in the saveLoginData function

```

def registerAccount():
    username=usernameBox.get()
    password=passwordBox.get()
    admin=adminBox.get()
    logindata={'Username':username,'Password':password,'Admin':admin}
    login.append(logindata)
    saveLoginData()

def saveLoginData():
    try:
        fileHandle=open('login.csv','r+')
        fileContent=fileHandle.read()
        print(fileContent)
        if fileContent.strip()=='':
            fileHandle.write('Username,Password,Admin\n')
        for item in login:
            fileHandle.write('{Username},{Password},{Admin}'.format(**item))
            fileHandle.write('\n')
        fileHandle.close()
    except OSError:
        print('Can\'t write to file')

```



I fix this by removing the first speech mark. **Otherwise, I would receive an error**

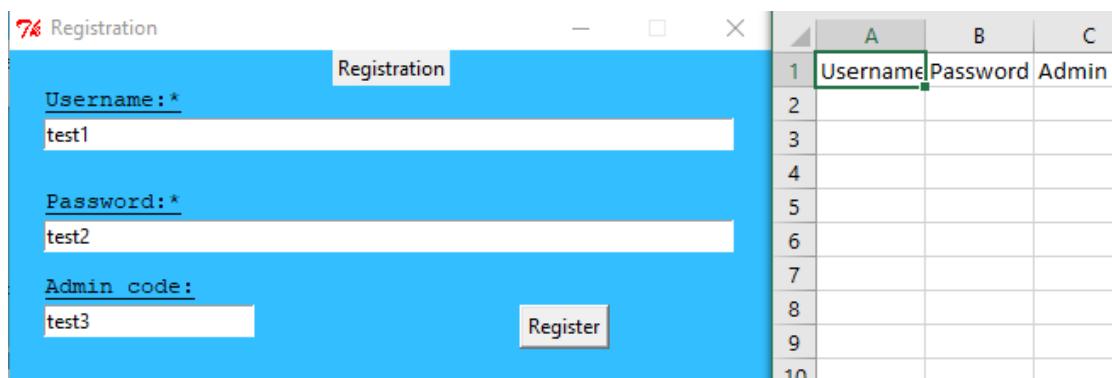
```

if fileContent.strip()=='':
    fileHandle.write('Username,Password,Admin\n')
for item in login:
    fileHandle.write('{Username},{Password},{Admin}\n')

```

## Testing

The program executes successfully. However, when the input boxes are filled and the register button is pressed, the login.csv file is not updated. **Without this, the user's detailed cannot be saved.**



## Error

I forgot to link the function to get the input from the boxes to the register button. I also receive a name error

```

buttonRegister=Button(registerWindow,text="Register",command=registerAccount)
buttonRegister.place(x=300,y=150)

```

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 36, in registerAccount
    username=usernameBox.get()
NameError: global name 'usernameBox' is not defined

```

## Error

I add parameters and arguments so the function can access the variables (**otherwise, I would receive an error**), but receive an unbound local error

```

def registerAccount(inputtedUsername, inputtedPassword, inputtedAdmin):
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    admin=inputtedAdmin.get()

buttonRegister=Button(registerWindow,text="Register",command=registerAccount(usernameBox,passwordBox,adminBox))
buttonRegister.place(x=300,y=150)

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 83, in registerFunction
    buttonRegister=Button(registerWindow,text="Register",command=registerAccount
(usernameBox,passwordBox,adminBox))
UnboundLocalError: local variable 'adminBox' referenced before assignment

```

## Error

I fix this by defining the adminBox variable before it is used. However, I now get a name error

```

adminBox=Entry(registerWindow,width=20)
adminBox.place(x=20,y=150)

buttonRegister=Button(registerWindow,text="Register",command=registerAccount(usernameBox,passwordBox,adminBox))
buttonRegister.place(x=300,y=150)

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 88, in registerFunction
    buttonRegister=Button(registerWindow,text="Register",command=registerAccount
(usernameBox,passwordBox,adminBox))
  File "N:\GUI (1).py", line 40, in registerAccount
    login.append(logindata)
NameError: global name 'login' is not defined

```

I fix this by making the array 'login' global (**otherwise, the array can't be accessed outside the function**), as it is created within the openLogin function, so it can't be used in the registerAccount function.

I will turn this global variable into a local variable at a later stage.

```

def openLogin():
    csvFile=open('login.csv')
    csvFileContent=(csvFile)
    next(csvFile)

    DatabaseList=[]

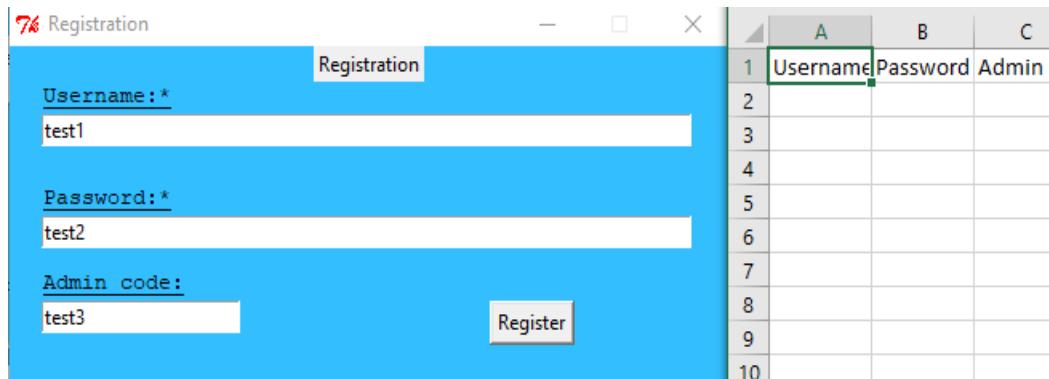
    for row in csvFileContent:
        database=row.strip().split(",")
        heading=['Username','Password','Admin']
        data=zip(heading,database)
        DataDict=dict(data)
        DatabaseList.append(DataDict)

    global login
    login=[]

```

## Testing

The program executes successfully again. However, when the input boxes are filled and the register button is pressed, the login.csv file is not updated. **Without this, the user's detailed cannot be saved.**



## Testing

I add some print functions to trace the execution of the functions (**otherwise, I would not know how my program executes**) and find that as soon as the button is pressed to open the registration window, all functions are executed.

```

def openLogin():
    csvFile=open('login.csv')
    csvFileContent=(csvFile)
    next(csvFile)

    DatabaseList=[]
    print("test1")

    for row in csvFileContent:
        database=row.strip().split(",")
        heading=['Username','Password','Admin']
        data=zip(heading,database)
        DataDict=dict(data)
        DatabaseList.append(DataDict)

    global login
    login=[]

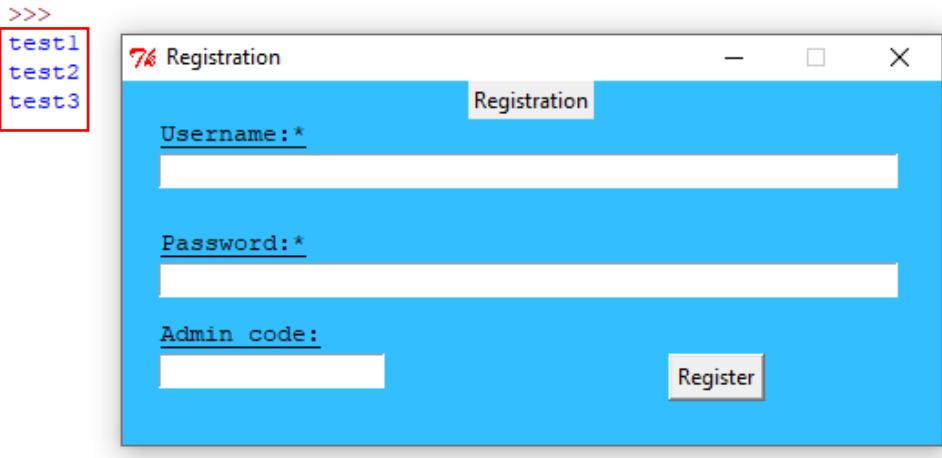
```

```

def registerAccount(inputtedUsername, inputtedPassword, inputtedAdmin):
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    admin=inputtedAdmin.get()
    print("test2")
    logindata={'Username':username,'Password':password,'Admin':admin}
    login.append(logindata)
    saveLoginData()

def saveLoginData():
    print("test3")
    try:
        fileHandle=open('login.csv','r+')
        fileContent=fileHandle.read()
        if fileContent.strip()=='':
            fileHandle.write('Username,Password,Admin\n')
        for item in login:
            fileHandle.write('{Username},{Password},{Admin}'.format(**item))
            fileHandle.write('\n')
        fileHandle.close()
    except OSError:
        print('Can\'t write to file')

```



While looking for the problem I realise both register buttons for the main menu and the registration menu have the same name. I change the name of the main menu register button, but this does not fix the problem.

```

buttonRegister=Button(registerWindow,text="Register",command=registerAccount(usernameBox,passwordBox,adminBox))
buttonRegister.place(x=300,y=150)

buttonRegistration=Button(root,text="Register a new\n account",command=registerFunction)
buttonRegistration.place(x=10, y=20)

```

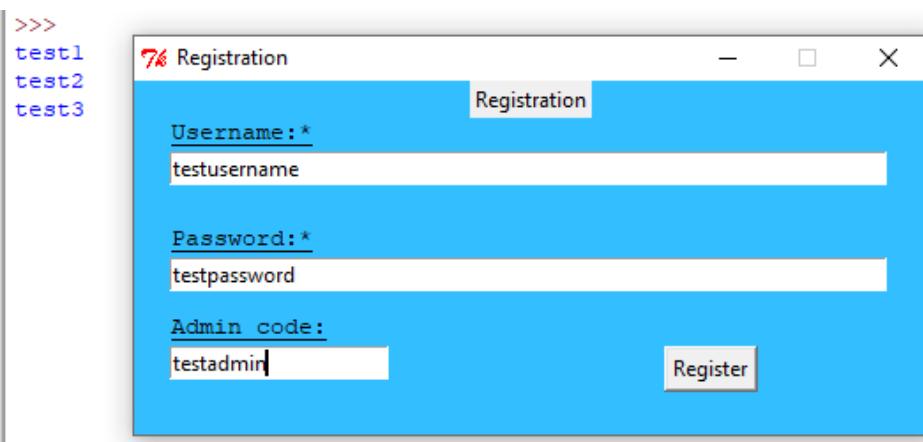
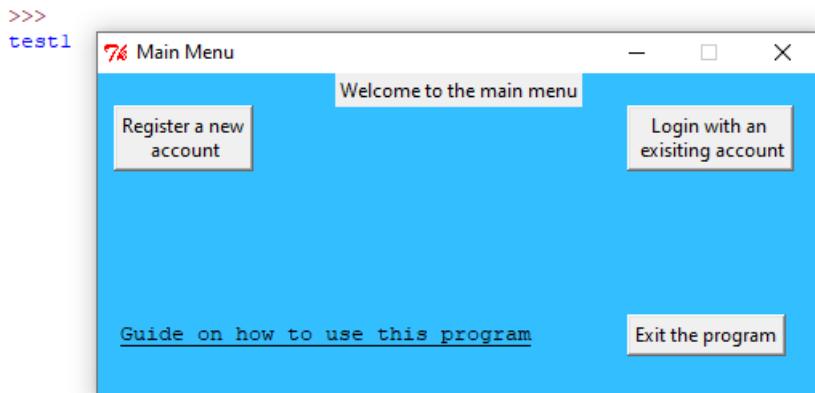
## Testing

To fix the problem, I use the 'lambda' function to hide the command until it is executed, **otherwise the functions are executed at the wrong time**. This prevents it from running before the button is pressed. This leads to only the openLogin function to be executed when the register option is picked from the main menu. Executing the registerAccount function along with the saveLoginData function within that, only when the register button is picked from the registration window. This causes the inputted boxes to be saved to the .csv file correctly

```

buttonRegister=Button(registerWindow,text="Register",command=lambda :registerAccount(usernameBox,passwordBox,adminBox))
buttonRegister.place(x=300,y=150)

```



Username	Password	Admin
testusername	testpassword	testadmin

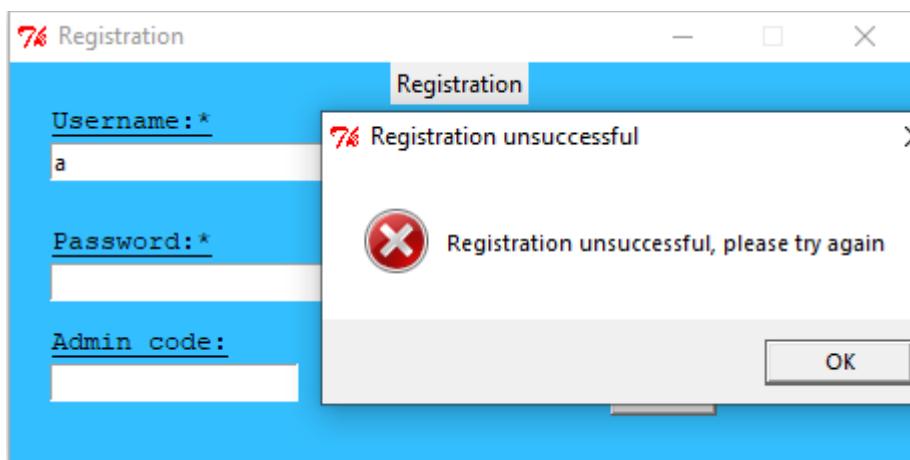
I also add a message box to indicate to the user that their registration was successful. **Otherwise, the user would not know if their account has been registered successfully.**

```
def saveLoginData():
    try:
        fileHandle=open('login.csv','r+')
        fileContent=fileHandle.read()
        if fileContent.strip()=='':
            fileHandle.write('Username,Password,Admin\n')
        for item in login:
            fileHandle.write('{Username},{Password},{Admin}'.format(**item))
            fileHandle.write('\n')
        fileHandle.close()
        messagebox.showinfo("Registration successful","Login details saved")
    except OSError:
        print('Can\'t write to file')
```



I also add an error message if the registration is not successful. **Otherwise, the user would not know if their account has been registered unsuccessfully.**

```
if username!="" and password!="":
    logindata={'Username':username,'Password':password,'Admin':admin}
    login.append(logindata)
    saveLoginData(login)
else:
    messagebox.showerror("Registration unsuccessful","Registration unsuccessful, please try again")
```



This is the end of my first prototype for registration menu. Next, I will create another window for logging in with an account and this will be linked to the 'login' button on the main menu.

## Login

### Login GUI

### Error

I create a login window based off my registration window (**otherwise, the user cannot enter their login details using the GUI**), but receive an error.

```

def loginFunction():
    openLogin()
    loginWindow=Tk()
    loginWindow.geometry("450x200")
    loginWindow.title("Login")
    loginWindow.configure(bg="#33BFFF")

    labelLogin=Label(loginWindow, text="Login")
    labelLogin.pack(side=TOP)

    loginWindow.resizable(False, False)

    labelUsername=Label(loginWindow, text="Username:", bg="#33BFFF", font=("Courier 10 underline"), borderwidth=0)
    labelUsername.place(x=20, y=20)

    usernameBox=Entry(loginWindow, width=67)
    usernameBox.place(x=20, y=40)

    labelPassword=Label(loginWindow, text="Password:", bg="#33BFFF", font=("Courier 10 underline"), borderwidth=0)
    labelPassword.place(x=20, y=80)

    passwordBox=Entry(loginWindow, width=67)
    passwordBox.place(x=20, y=100)

    buttonLogin=Button(loginWindow, text="Login")
    buttonLogin.place(x=300, y=150)

>>>
Traceback (most recent call last):
  File "N:\GUI (1).py", line 129, in <module>
    buttonLogin=Button(root, text="Login with an\nexisting account", command=LoginFunction)
NameError: name 'LoginFunction' is not defined
>>> |

```

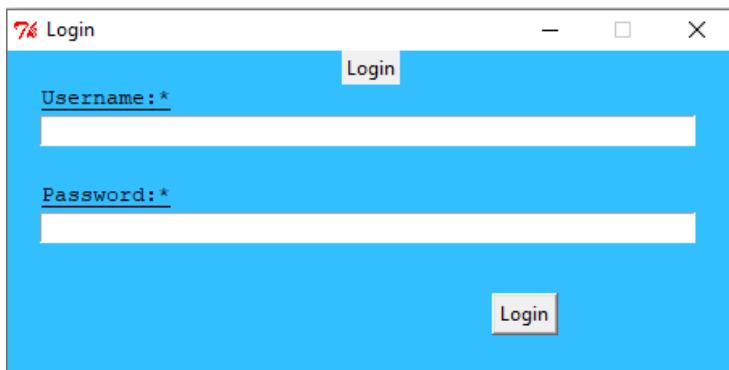
I fix this by changing the command to match the function name. **Otherwise, an error would appear**

```

def loginFunction():

    buttonLogin=Button(root, text="Login with an\nexisting account", command=LoginFunction)
    buttonLogin.place(x=330, y=20)

```

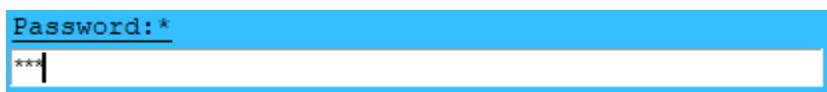


I also use the 'show' parameter to display asterisk when inputting a password and I also add this to my register function. **Otherwise, the security of the program is reduced as an inputted password is easily visible.**

```

passwordBox=Entry(loginWindow, width=67, show="*")
passwordBox.place(x=20, y=100)

```



I pack the login button instead of placing it. This correctly aligns the login button. **Otherwise, the login button is out of place.**

```
buttonLogin=Button(loginWindow, text="Login")
buttonLogin.pack(side=BOTTOM, pady=10)
```



## Retrieve and verify login details function

### Error

I try to pass the input boxes into a function (**without this the inputs cannot be accessed by this function**) ,but receive an error.

```
def loginAccount(inputtedUsername, inputtedPassword, inputtedAdmin):
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    print(username)
    print(password)
```

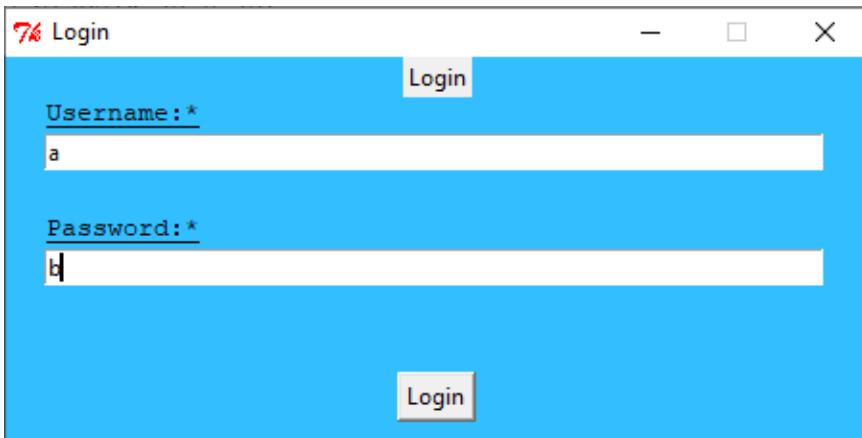
```
buttonLogin=Button(root, text="Login with an\nexisting account", command=lambda :loginAccount(usernameBox, passwordBox))
```

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 130, in <lambda>
    buttonLogin=Button(root, text="Login with an\nexisting account", command=lam
bda :loginAccount(usernameBox, passwordBox))
NameError: global name 'usernameBox' is not defined
```

I fix this by moving this command to the login button inside the login window instead of the login button inside of the main menu, **otherwise I would receive an error**. Also, both inputted values are printed, showing that the inputs are passed through the function.

```
buttonLogin=Button(loginWindow, text="Login", command=lambda :loginAccount(usernameBox, passwordBox))
buttonLogin.pack(side=BOTTOM, pady=10)
```

```
buttonLogin=Button(root, text="Login with an\nexisting account", command=loginFunction)
buttonLogin.place(x=330, y=20)
```



(program output)

```
>>>
a
b
```

## Error

I create a simple login function to test some variables but receive a global error.

```
def loginAccount(inputtedUsername, inputtedPassword):
    openLogin()
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    for i in DatabaseList:
        if i['Username']==username and i['Password']==password:
            print("Login successful")
            if username['Admin']=='testAdmin':
                print("test")
                #OPEN SECONDARY MENU
                break
        else:
            print("Login unsuccessful, please check your login details again")
            loginAccount(username,password)

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 118, in <lambda>
    buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(use
rnameBox,passwordBox))
  File "N:\GUI (1).py", line 125, in loginAccount
    for i in DatabaseList:
NameError: global name 'DatabaseList' is not defined
```

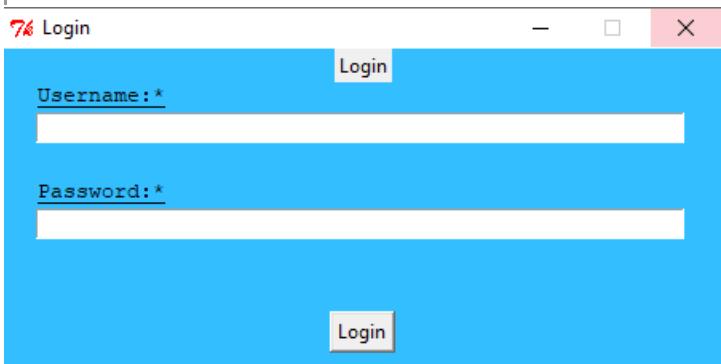
I fix this by making the array 'DatabaseList' global, so it can be used in the function. **Otherwise, the array would be inaccessible in this function.**

I will make this a local array at a later stage.

```
def openLogin():
    csvFile=open('login.csv')
    csvFileContent=(csvFile)
    next(csvFile)

    global DatabaseList
    DatabaseList=[]
```

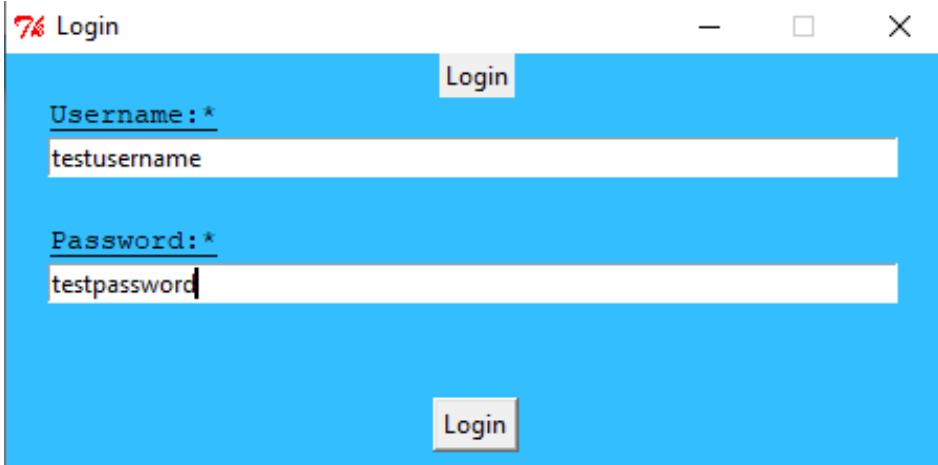
DatabaseList made  
global to prevent  
error



## Error

When entering correct login details (that also contains an admin code), the username and password are seen as correct, however the program fails to verify the admin code. **Without this verification, the security of the program is reduced.**

Username	Password	Admin
testusername	testpassword	testadmin
noadmin1	noadmin2	



```
>>>
Login successful
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 119, in <lambda>
    buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(usernameBox,passwordBox))
  File "N:\GUI (1).py", line 129, in loginAccount
    if username['Admin']=='testAdmin':
TypeError: string indices must be integers
```

I fix this by correcting the searching of the 'admin' column. Otherwise, the program cannot correctly verify the privileges of a user's account.

```
if i['Admin']=='testadmin':
    print("test")
    #OPEN SECONDARY MENU
break
```

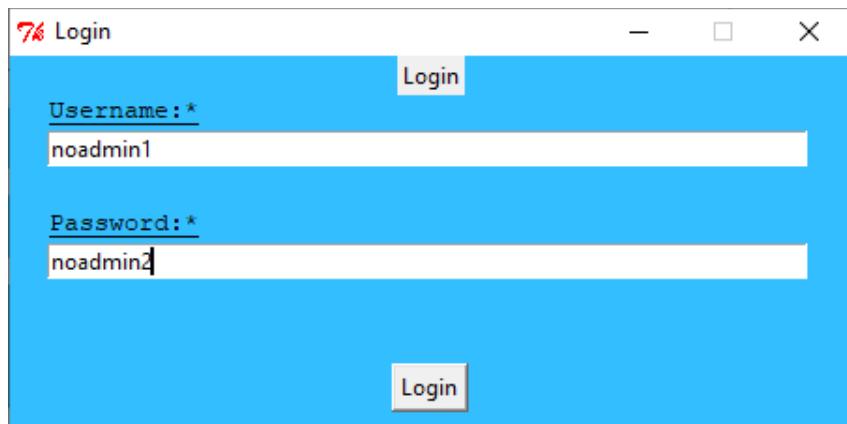
(program output)

```
>>>
Login successful
test
```

## Error

When entering correct login details (that does not contain an admin code), the username and password are not seen as correct. An error message appears.

Username	Password	Admin
testusername	testpassword	testadmin
noadmin1	noadmin2	

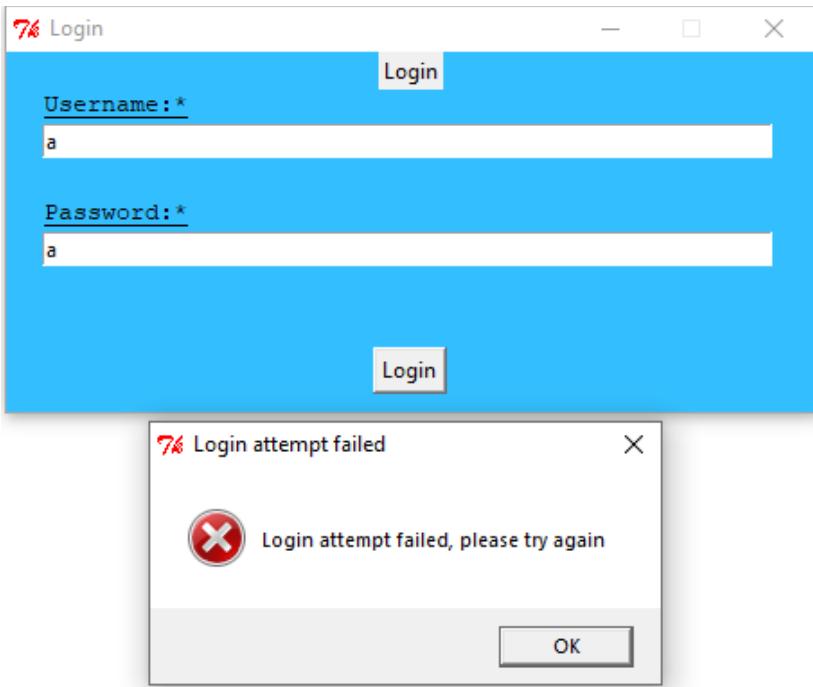


```
>>>
Login unsuccessful, please check your login details again
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 119, in <lambda>
    buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(usernameBox,passwordBox))
  File "N:\GUI (1).py", line 135, in loginAccount
    loginAccount(username,password)
  File "N:\GUI (1).py", line 124, in loginAccount
    username=inputtedUsername.get()
AttributeError: 'str' object has no attribute 'get'
```

I realise a recursion is not needed to allow the user to re-enter their login details, as the process of checking the details always is available by pressing the login button. Therefore, I remove the last part of the loginAccount function, replacing it with an alert to let the user know their login attempt failed and to try again. **Otherwise, the user would not know if their login attempt was unsuccessful.**

```
def loginAccount(inputtedUsername,inputtedPassword):
    openLogin()
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    loginStatus=False
    for i in DatabaseList:
        if i['Username']==username and i['Password']==password:
            print("Login successful")
            loginStatus=True
            if i['Admin']=='testadmin':
                print("test")
                #OPEN SECONDARY MENU
                break
        break
    if loginStatus==False:
        messagebox.showerror("Login attempt failed","Login attempt failed, please try again")|
```

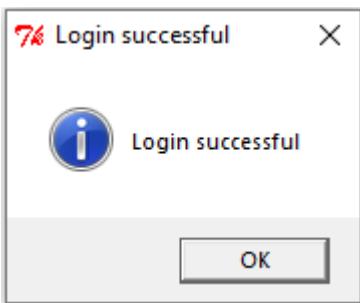
Username	Password	Admin
testusername	testpassword	testadmin
noadmin1	noadmin2	



I also add a message box to indicate to the user that their login was successful. **Otherwise, the user would not know if their login attempt was successful.**

```
for i in databaseList:
    if i['Username']==username and i['Password']==password:
        messagebox.showinfo("Login successful","Login successful")
```

Message box added. **Otherwise, the user does not know if their login attempt is successful**



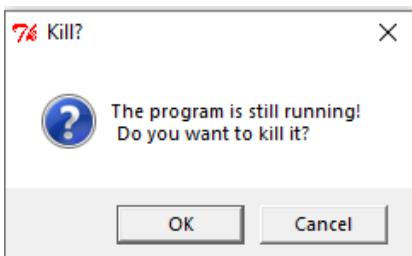
This is the end of my first prototype for registration menu. Next, I create the exit for the program.

## Exit

### Exit button functionality

I first try to use the built-in exit command. However, this creates a warning prompt, and I would like the program to shut down instantly. **Otherwise, the button is not efficient in closing the program**

```
buttonEnd=Button(root,text="Exit the program",command=exit)
buttonEnd.place(x=330,y=150)
```



Next, I import the sys module and try the sys.exit command. **Without this, I would not be able to implement an exit button.**

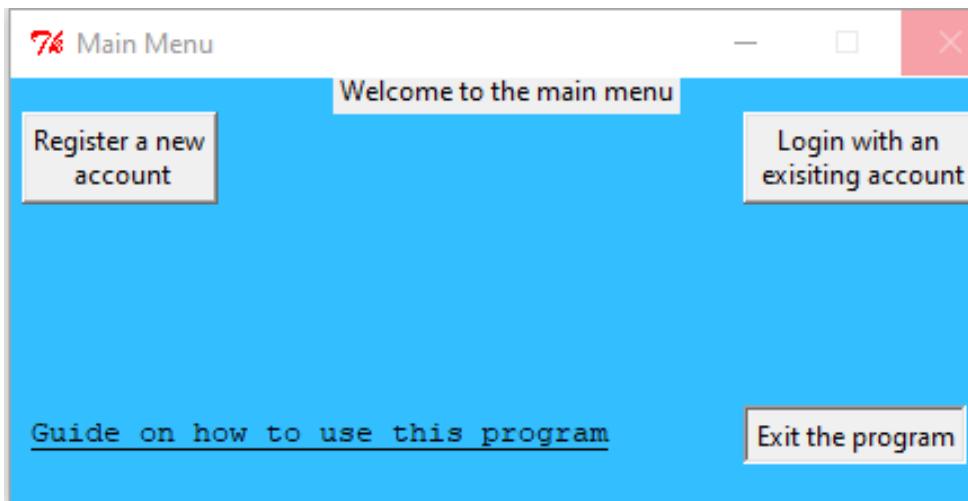
```
from tkinter import *
import csv
import sys
```

sys module imported.  
Otherwise, I cannot use  
the module's functions

```
buttonEnd=Button(root,text="Exit the program",command=sys.exit)
buttonEnd.place(x=330,y=150)
```

## Error

This causes the program to freeze and display an error message, before crashing.



```
>>>
Traceback (most recent call last):
  File "N:\GUI (1).py", line 154, in <module>
    root.mainloop()
  File "C:\Python32\lib\tkinter\__init__.py", line 1009, in mainloop
    self.tk.mainloop(n)
  File "C:\Python32\lib\tkinter\__init__.py", line 1401, in __call__
    raise SystemExit(msg)
SystemExit
```

Then, I import the os module and use the os.\_exit command, **Without this, I would not be able to implement an exit button.** But, the program fails to run.

```
from tkinter import *
import csv
import os
```

os module imported.  
**Otherwise, I cannot use the module's functions**

```
buttonEnd=Button(root,text="Exit the program",command=os._exit(0))
buttonEnd.place(x=330,y=150)
```

To fix this I use the lambda function this prevents the command from executing instantly, which allows the program to run. **Without this, the program will not run making it unusable.** The os.\_exit command also works in instantly closing the program.

```
buttonEnd=Button(root,text="Exit the program",command=lambda: os._exit(0))
buttonEnd.place(x=330,y=150)
```

This is the end of my first prototype for the exit button. Next, I will create a guide on how to use the program that is accessible from the main menu

## Guide

### Open and read guide function

First, I create a .txt file in the same folder as the program. I will add a guide on the functionality so far and continue adding to it as I develop my program. I will create a function to read the text in the file. **Otherwise, the guide cannot be seen using the program.**

 GUI (1)	29/11/2022 13:43	Python File	5 KB
 guide	30/11/2022 11:35	Text Document	0 KB
 login	28/11/2022 10:27	Microsoft Excel C...	1 KB

I create a function to open and read the text file and link this function to the button's command. **Otherwise, the guide cannot be seen using the program.**

```
def openGuide():
    txtFile=open('guide.txt')
    print(txt.read())
    +
buttonGuide=Button(root,text="Guide on how to use this program",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0, command=openGuide)
buttonGuide.place(x=10,y=150)
```

## Error

I get a name error and fix it by correctly naming the variable.

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 147, in openGuide
    print(txt.read())
NameError: global name 'txt' is not defined
```

```
def openGuide():
    txtFile=open('guide.txt')
    print(txtFile.read())
```

Variable name  
was incorrect

Pressing the button successfully prints the file contents to the shell of the program. However, to make this easier to read I print the file contents to a separate window. **Otherwise, the guide cannot be seen using the GUI.**

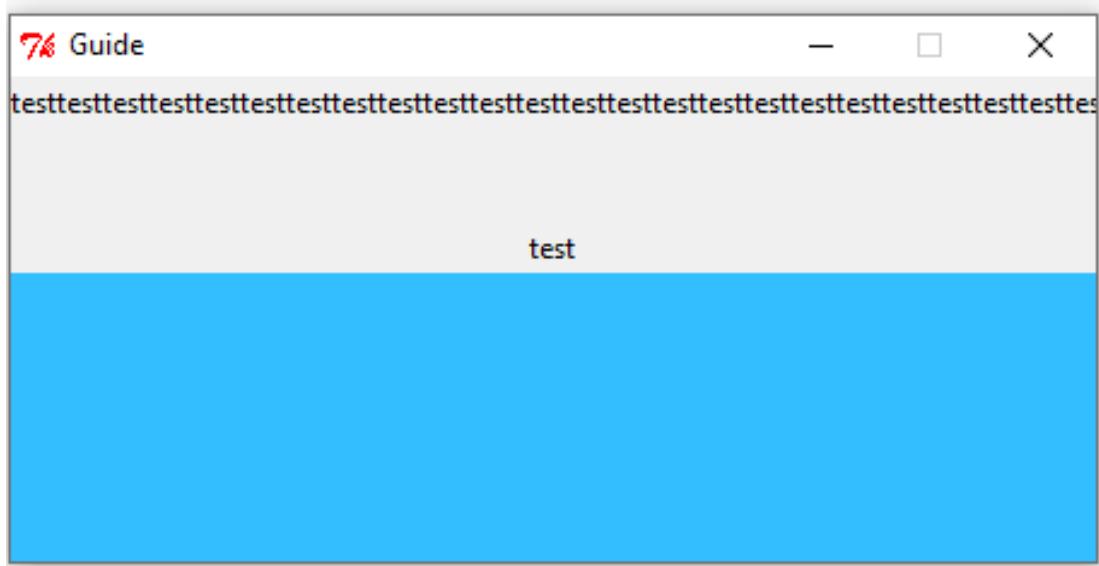
## Guide GUI

```
""
def openGuide():
    guideWindow=Tk()
    guideWindow.geometry("450x200")
    guideWindow.title("Guide")
    guideWindow.configure(bg="#33BFFF")

    guideWindow.resizable(False,False)

    txtFile=open('guide.txt')

    labelGuide=Label(guideWindow,text=txtFile.read())
    labelGuide.pack()
```



I change the background of the label to match the window background. **Otherwise, the text does not match the design of the window.**

```
def openGuide():
    guideWindow=Tk()
    guideWindow.geometry("450x200")
    guideWindow.title("Guide")
    guideWindow.configure(bg="#33BFFF")

    guideWindow.resizable(False, False)

    txtFile=open('guide.txt')

    labelGuide=Label(guideWindow, text=txtFile.read(), bg="#33BFFF")
    labelGuide.pack()
```



I realise the words do not wrap around to the next line proportional to the size of the window. My initial solution was to allow the window to be resizable, but this does not position the text correctly.

```
def openGuide():
    guideWindow=Tk()
    guideWindow.geometry("450x200")
    guideWindow.title("Guide")
    guideWindow.configure(bg="#33BFFF")

    #guideWindow.resizable(False,False)

    txtFile=open('guide.txt')

    labelGuide=Label(guideWindow,text=txtFile.read(),bg="#33BFFF")
    labelGuide.pack()
```



Next, I try the width parameter to limit the size of the label, but this cuts off the text.

```
def openGuide():
    guideWindow=Tk()
    guideWindow.geometry("450x200")
    guideWindow.title("Guide")
    guideWindow.configure(bg="#33BFFF")

    #guideWindow.resizable(False, False)

    txtFile=open('guide.txt')

    labelGuide=Label(guideWindow,text=txtFile.read(),bg="#33BFFF", width=50)
    labelGuide.pack()
```

width parameter used to attempt to make all the text visible on the window. **Otherwise, the user cannot read the guide file**



Next, I use the wraplength parameter to limit the characters in each line. **Otherwise, the full contents of the guide would not be visible.**

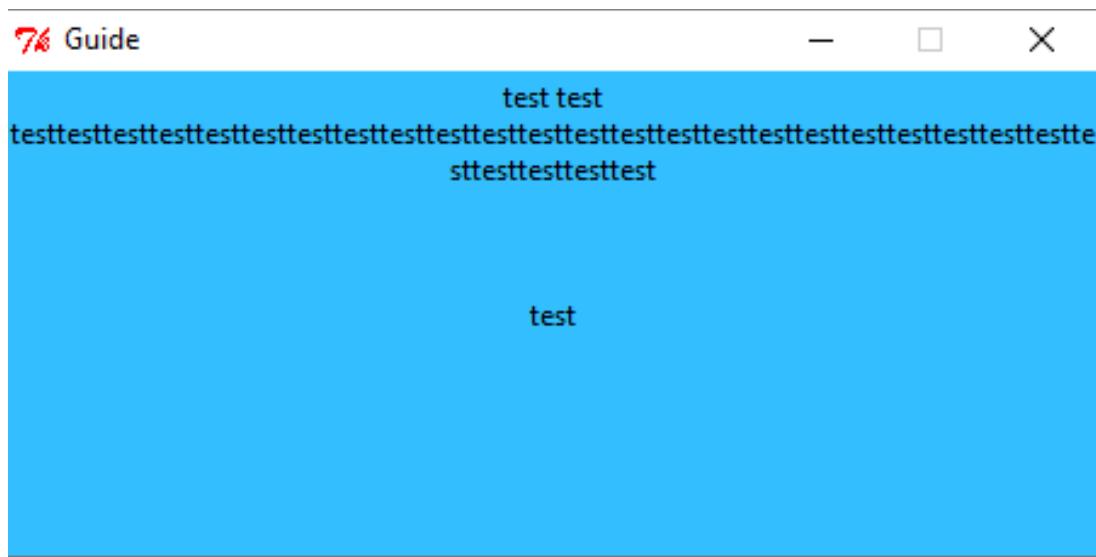
```
def openGuide():
    guideWindow=Tk()
    guideWindow.geometry("450x200")
    guideWindow.title("Guide")
    guideWindow.configure(bg="#33BFFF")

    guideWindow.resizable(False, False)

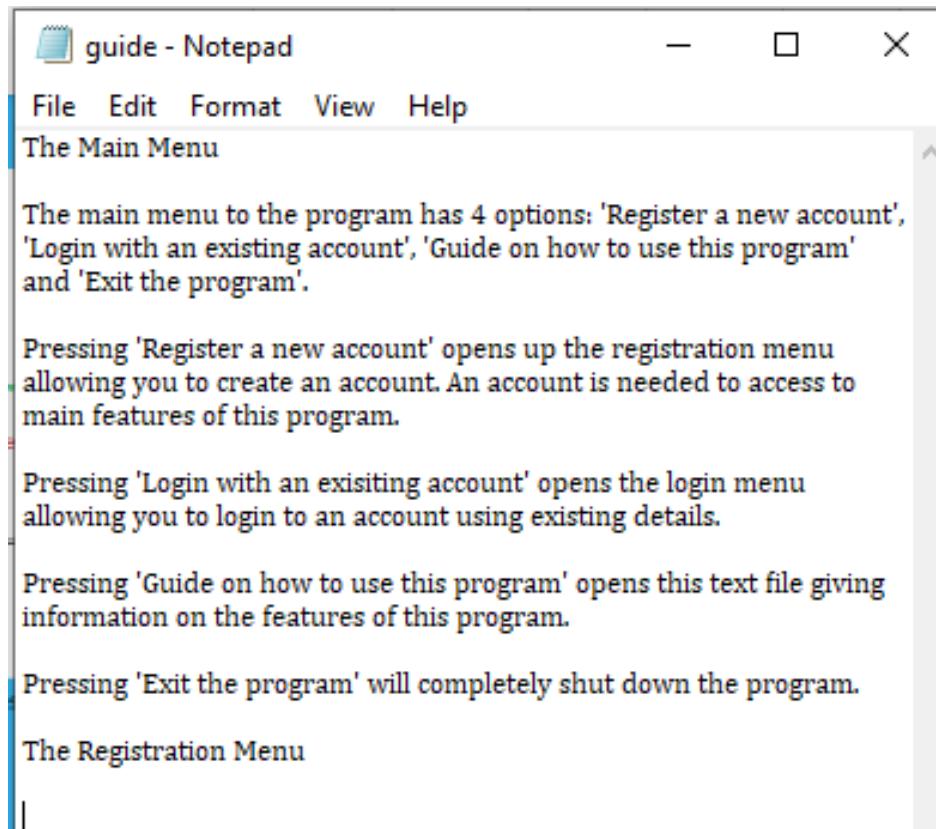
    txtFile=open('guide.txt')

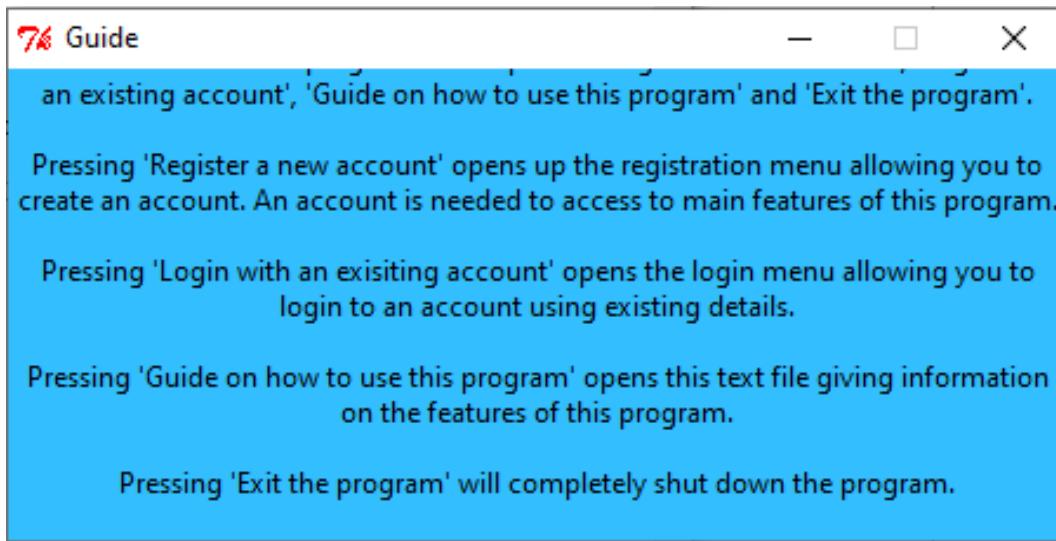
    labelGuide=Label(guideWindow, text=txtFile.read(), bg="#33BFFF", wraplength=450)
    labelGuide.pack()
```

wraplength parameter used  
to limit the characters in  
each line. **Otherwise, the  
full contents of the guide  
would not be visible.**



After writing some information, I realise the window is too small to fit all the text vertically. To fix this I will implement a scroll bar to view everything. **Otherwise, the full contents of the guide would not be visible.**





## Error

I use tkinter's 'Scrollbar' function to create a scroll bar and link this to the label holding all the text (**without this the guide would not be fully visible**), but receive an error

```
labelGuide=Label(guideWindow,text=txtFile.read(),bg="#33BFFF", wraplength=450,yscrollcommand=scrollbar.set)
labelGuide.pack()

scrollbar=Scrollbar(guideWindow)
scrollbar.pack(side=RIGHT)

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 155, in openGuide
    labelGuide=Label(guideWindow,text=txtFile.read(),bg="#33BFFF", wraplength=45
0,yscrollcommand=scrollbar.set)
UnboundLocalError: local variable 'scrollbar' referenced before assignment
```

## Error

To fix this I define the scrollbar before linking it to the label, but I receive another error

```
scrollbar=Scrollbar(guideWindow)
scrollbar.pack(side=RIGHT)

labelGuide=Label(guideWindow,text=txtFile.read(),bg="#33BFFF", wraplength=450,yscrollcommand=scrollbar.set)
labelGuide.pack()

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI (1).py", line 158, in openGuide
    labelGuide=Label(guideWindow,text=txtFile.read(),bg="#33BFFF", wraplength=45
0,yscrollcommand=scrollbar.set)
  File "C:\Python32\lib\tkinter\__init__.py", line 2459, in __init__
    Widget.__init__(self, master, 'label', cnf, kw)
  File "C:\Python32\lib\tkinter\__init__.py", line 1958, in __init__
    (widgetName, self._w) + extra + self._options(cnf))
tkinter.TclError: unknown option "-yscrollcommand"
```

I find out that a scroll bar can only be added to certain widgets. To fix this I change my label widget to a text widget, **otherwise I cannot implement the scroll bar**. I also configure the scrollbar so that when it is added, it can interact with the window.

```
txtFile=open('guide.txt')

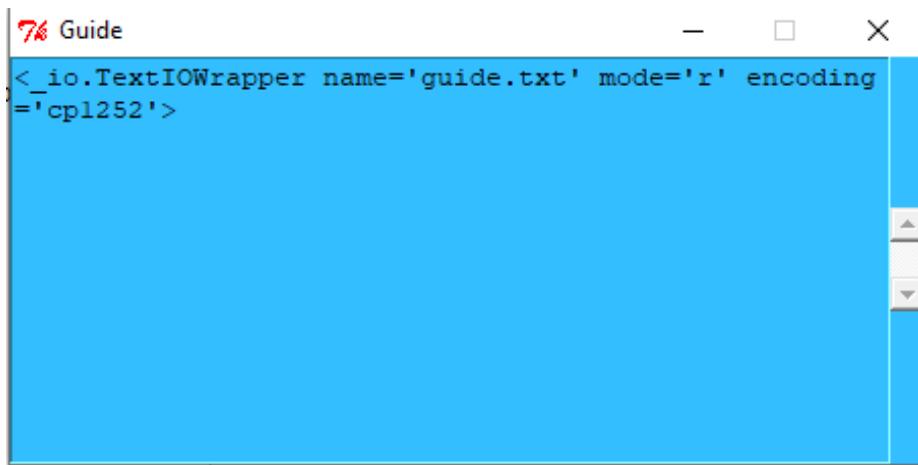
scrollbar=Scrollbar(guideWindow)
scrollbar.pack(side=RIGHT)

textGuide=Text(guideWindow,bg="#33BFFF", width=450)
textGuide.pack()

textGuide.insert(END,txtFile)

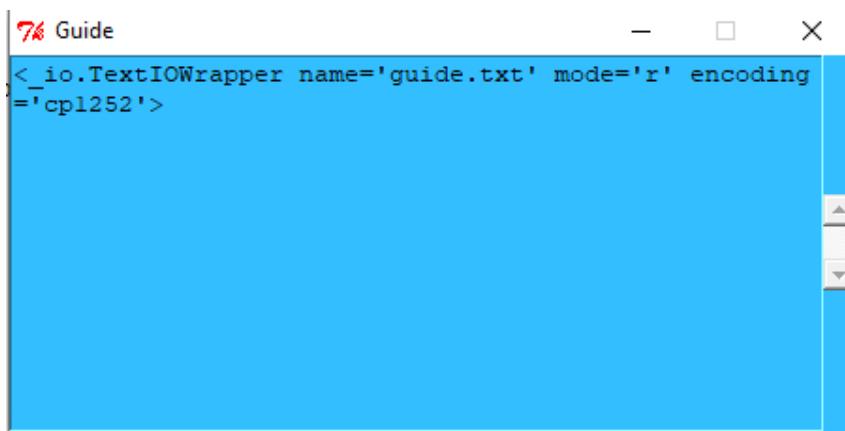
textGuide.config(yscrollcommand=scrollbar.set)
scrollbar.config(command=textGuide.yview)
```

text widget used as it is compatible with the scrollbar. **Otherwise, the scrollbar cannot be implemented**



The scroll bar is implemented, but the text is not displaying properly. To fix this problem, I attempt to open the text file in the read mode. However, this does not fix the problem.

```
txtFile=open('guide.txt','r')
```



To fix this I read the contents of the file to a variable (text), which I then insert into the text widget. **Otherwise, the guide is not visible on the GUI.**

```

textFile=open('guide.txt')
text=textFile.read()

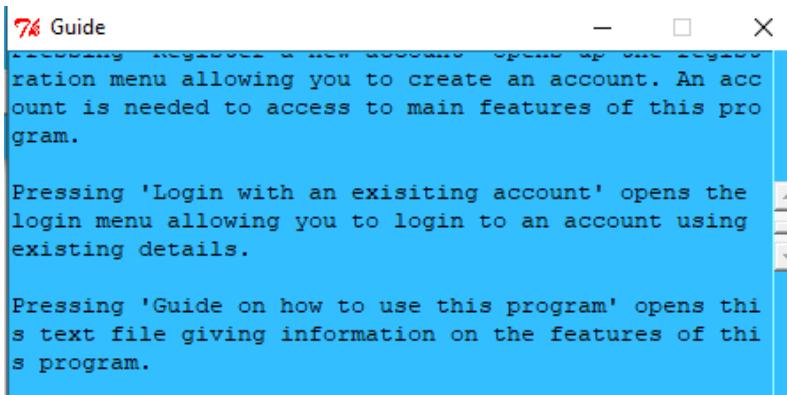
scrollbar=Scrollbar(guideWindow)
scrollbar.pack(side=RIGHT)

textGuide=Text(guideWindow,bg="#33BFFF", width=450)
textGuide.pack()

textGuide.insert(END, text)

textGuide.config(yscrollcommand=scrollbar.set)
scrollbar.config(command=textGuide.yview)

```

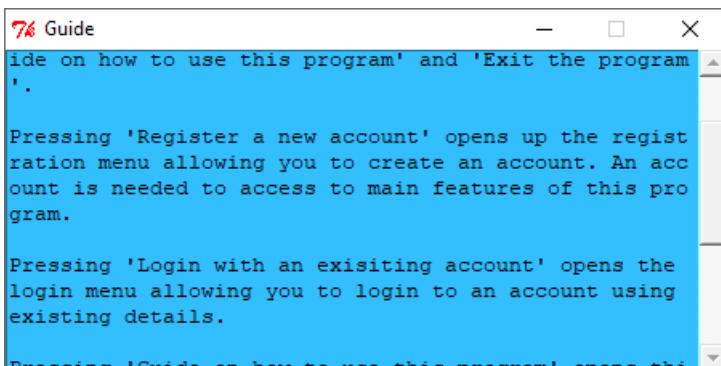


To fix the scrollbar, I use the 'fill' parameter when packing the scrollbar. This allows the scrollbar to fill the rest of the vertical space. **Otherwise, the scroll bar is hard to use and does not match the design of the window.**

```

scrollbar=Scrollbar(guideWindow)
scrollbar.pack(side=RIGHT, fill=Y)

```

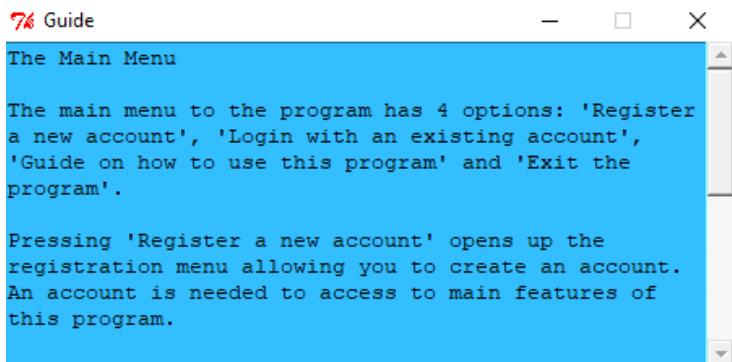


To make the text easier to read, I use the 'wrap' parameter for the text widget. This breaks the line after the last word if a sentence is too long, compared to breaking after the last character. **Without this, the user would find it difficult to read the guide.**

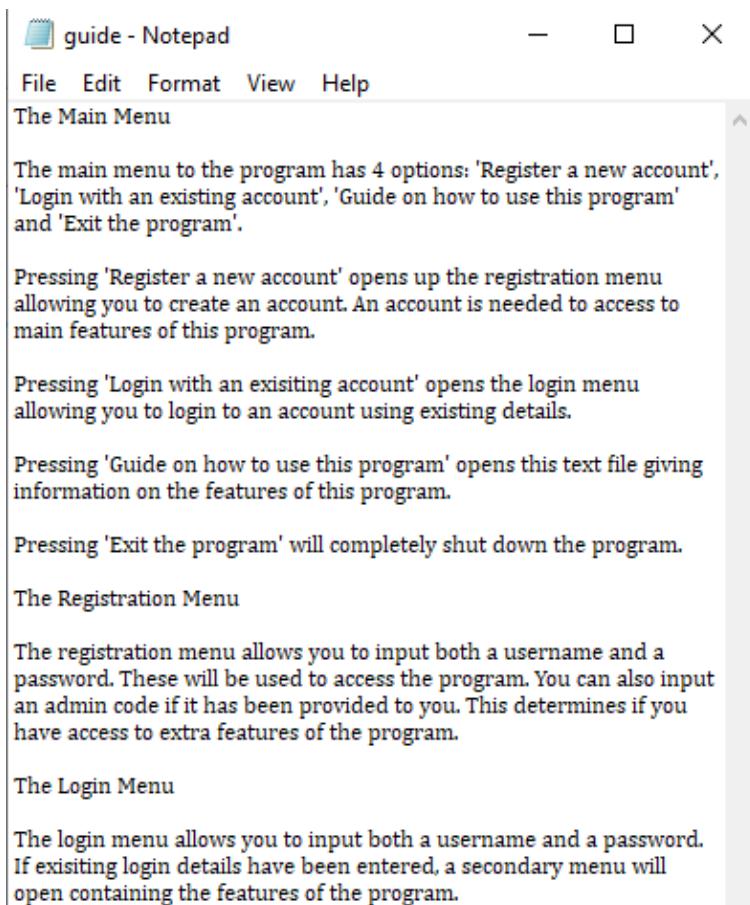
```

textGuide=Text(guideWindow,bg="#33BFFF", width=450, wrap=WORD)
textGuide.pack()

```



Then, I add some extra details to the guide. **Otherwise, a user will find it hard to learn from the guide.**

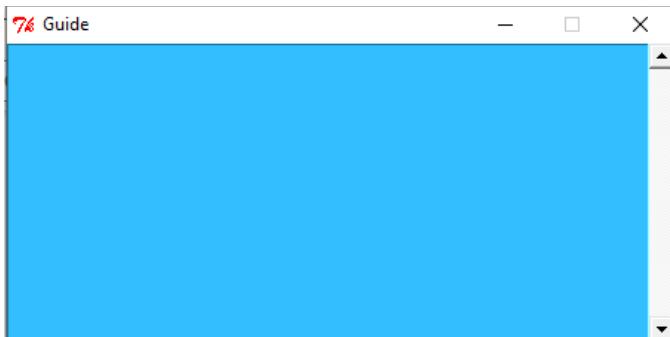


## Testing

When checking the guide appears correctly, I learn that the text widget can be modified by the user when the program is running. I try to fix this using the 'state' parameter, which should prevent the widget from responding to mouse and keyboard events, **otherwise the user can modify the text**. However, this prevents any text from showing.

```
textGuide=Text (guideWindow, bg="#33BFFF", width=450, wrap=WORD, state=DISABLED)
textGuide.pack()
```

state parameter used to prevent the user from modifying the guide

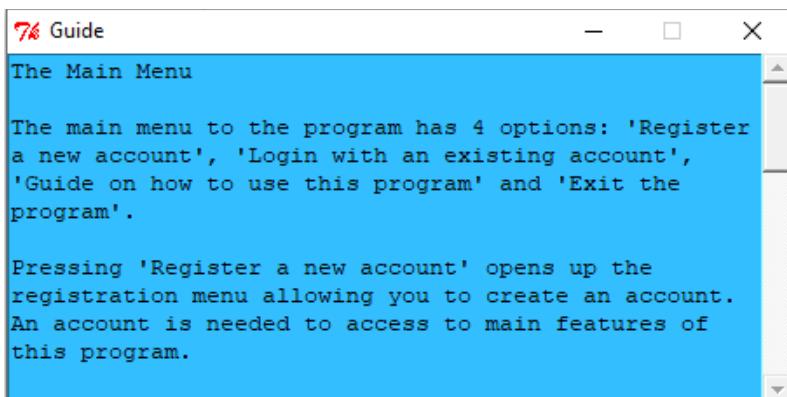


To fix this, I use the 'state' parameter in the config section instead of when defining the text widget. Due to the order the program reads the code, this allows the text to be inserted into the widget then the state is set to disabled then the window is displayed to the user, preventing the user from modifying the text.

```
textGuide.insert(END, text)

textGuide.config(yscrollcommand=scrollbar.set, state=DISABLED)
scrollbar.config(command=textGuide.yview)
```

state parameter used to  
prevent the user from  
modifying the guide



This is the end of my first prototype for guide button and the guide itself. Next, I will create the secondary menu that will appear when a correct login is inputted into the login menu.

## Secondary Menu

### Secondary Menu GUI

First, I create a second window for the secondary menu. **Otherwise, key features of the program could not be accessed.**

```
#  
#      SECONDARY MENU  
  
#  
  
sec=Tk()  
sec.geometry("450x200")  
sec.title("Features Menu")  
sec.configure(bg="#33BFFF")  
  
sec.resizable(False,False)  
  
labelSecMenu=Label(sec,text="Welcome to the program")  
labelSecMenu.pack(side=TOP)
```

## Error

Then, I link this to execute when a correct login is entered, but I get an error.

```
def loginAccount(inputtedUsername,inputtedPassword):  
    openLogin()  
    username=inputtedUsername.get()  
    password=inputtedPassword.get()  
    loginStatus=False  
    for i in DatabaseList:  
        if i['Username']==username and i['Password']==password:  
            messagebox.showinfo("Login successful","Login successful")  
            sec.mainloop()  
            loginStatus=True  
            if i['Admin']=='testadmin':  
                print("test")  
                break  
            break  
    if loginStatus==False:  
        messagebox.showerror("Login attempt failed","Login attempt failed, please try again")  
  
Exception in Tkinter callback  
Traceback (most recent call last):  
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__  
    return self.func(*args)  
  File "N:\GUI.py", line 124, in <lambda>  
    buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(userNameBox,passwordBox))  
  File "N:\GUI.py", line 135, in loginAccount  
    sec.mainloop()  
NameError: global name 'sec' is not defined
```

To fix this, I create the second window at the beginning of the program, so I can call and add to it throughout the rest of the program. However, the secondary menu now opens when the program is run.

```
from tkinter import *
import csv
import os

#
#  MAIN MENU
#

root=Tk()
root.geometry("450x200")
root.title("Main Menu")
root.configure(bg="#33BFFF")

root.resizable(False,False)

labelMenu=Label(root,text="Welcome to the main menu")
labelMenu.pack(side=TOP)

#
#  SECONDARY MENU
#

sec=Tk()
sec.geometry("450x200")
sec.title("Features Menu")
sec.configure(bg="#33BFFF")

sec.resizable(False,False)

labelSecMenu=Label(sec,text="Welcome to the program")
labelSecMenu.pack(side=TOP)
```



Secondary menu is created at the start of the program now

Only the main menu window should open when the program is first run. However, the features menu also opens

To fix this I create the secondary menu in the login function. This makes sense as it can only be accessed when logging in. However, I will find a way to create it in the main program if this causes problems later in the program.

```
def loginAccount(inputtedUsername, inputtedPassword):
    sec=TK()
    sec.geometry("450x200")
    sec.title("Features Menu")
    sec.configure(bg="#33BFFF")

    sec.resizable(False, False)

    labelSecMenu=Label(sec, text="Welcome to the program")
    labelSecMenu.pack(side=TOP)

    openLogin()
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    loginStatus=False
    for i in DatabaseList:
        if i['Username']==username and i['Password']==password:
            messagebox.showinfo("Login successful", "Login successful")
            sec.mainloop()
            loginStatus=True
            if i['Admin']=="testadmin":
                print("test")
                break
        break
    if loginStatus==False:
        messagebox.showerror("Login attempt failed", "Login attempt failed, please try again")
```

GUI defined in the function so that it only opens when the function is called, instead of when the program is ran

As a better solution, I create the secondary menu in a function that is defined at the beginning of the program. During the login, if existing login details are entered the function is executed, which also opens the window.  
**Otherwise, after a successful login the secondary menu is not opened.**

```

from tkinter import *
import csv
import os

#
#    MAIN MENU
#

root=Tk()
root.geometry("450x200")
root.title("Main Menu")
root.configure(bg="#33BFFF")

root.resizable(False,False)

labelMenu=Label(root,text="Welcome to the main menu")
labelMenu.pack(side=TOP)

#
#    SECONDARY MENU
#
def secondaryMenu():
    sec=Tk()
    sec.geometry("450x200")
    sec.title("Features Menu")
    sec.configure(bg="#33BFFF")

    sec.resizable(False,False)

    labelSecMenu=Label(sec,text="Welcome to the program")
    labelSecMenu.pack(side=TOP)

def loginAccount(inputtedUsername,inputtedPassword):
    openLogin()
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    loginStatus=False
    for i in DatabaseList:
        if i['Username']==username and i['Password']==password:
            messagebox.showinfo("Login successful","Login successful")
            secondaryMenu()
            loginStatus=True
            if i['Admin']=='testadmin':
                print("test")
                break
            break
    if loginStatus==False:
        messagebox.showerror("Login attempt failed","Login attempt failed, pleas

```

Secondary menu  
defined in a seperate  
function so that it can  
be executed only when  
the user successfully  
logs in

Next, I add the buttons for all the features of the program. **Otherwise, the user could not access the options.**

```
def secondaryMenu():
    sec=Tk()
    sec.geometry("450x200")
    sec.title("Features Menu")
    sec.configure(bg="#33BFFF")

    sec.resizable(False, False)

    labelSecMenu=Label(sec, text="Welcome to the program")
    labelSecMenu.pack(side=TOP)

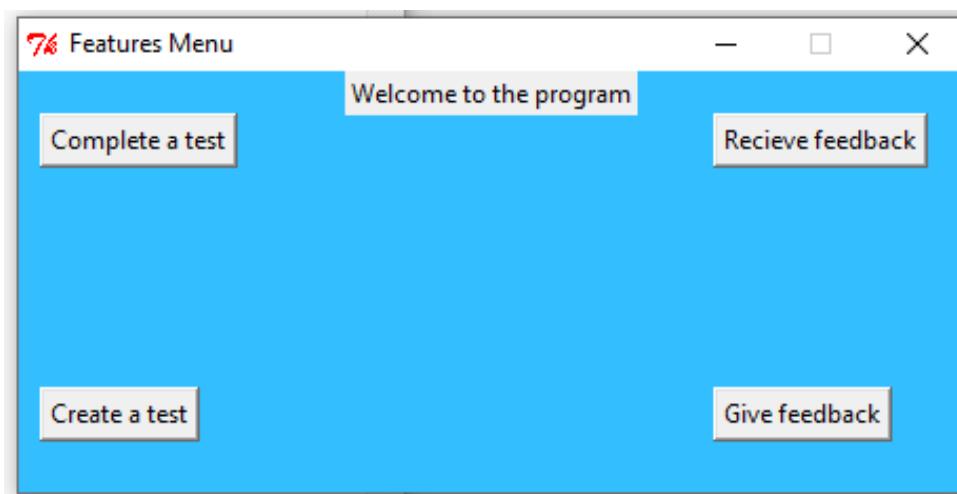
    buttonComplete=Button(sec, text="Complete a test")
    buttonComplete.place(x=10, y=20)

    buttonRFeedback=Button(sec, text="Recieve feedback")
    buttonRFeedback.place(x=330, y=20)

    buttonGFeedback=Button(sec, text="Give feedback")
    buttonGFeedback.place(x=330, y=150)

    buttonCreate=Button(sec, text="Create a test")
    buttonCreate.place(x=10, y=150)
```

Buttons and labels added to the window. **Otherwise, the user would not be able to access the different options, and the user would not know what the buttons do.** without the labels.



Then, I make the admin features only appear if your account has the correct admin code. **Otherwise, any user can access the admin features, this would be a security risk.**

```

def loginAccount(inputtedUsername, inputtedPassword):
    openLogin()
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    loginStatus=False
    global adminStatus
    adminStatus=False
    for i in DatabaseList:
        if i['Username']==username and i['Password']==password:
            messagebox.showinfo("Login successful", "Login successful")
            loginStatus=True
            if i['Admin']=="testadmin":
                adminStatus=True
    if loginStatus==True:
        secondaryMenu()
    else:
        messagebox.showerror("Login attempt failed", "Login attempt failed, please try again")

```



I also limit the size of the screen if a non-admin account is used as there is a lot of empty space. **Otherwise, the window takes up more space than necessary.**

```

def secondaryMenu(adminStatus):
    sec=Tk()
    sec.geometry("450x50")
    sec.title("Features Menu")
    sec.configure(bg="#33BFFF")

    sec.resizable(False, False)

    labelSecMenu=Label(sec, text="Welcome to the program")
    labelSecMenu.pack(side=TOP)

    buttonComplete=Button(sec, text="Complete a test")
    buttonComplete.place(x=10, y=20)

    #
    # GUI with input box to enter test name
    #

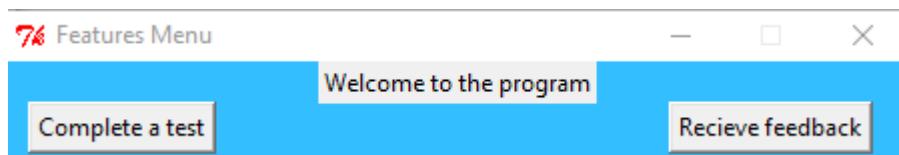
    buttonRFeedback=Button(sec, text="Recieve feedback")
    buttonRFeedback.place(x=330, y=20)

    if adminStatus==True:
        sec.geometry("450x200")

        buttonGFeedback=Button(sec, text="Give feedback")
        buttonGFeedback.place(x=330, y=150)

```

The window is made smaller if the user does not have admin status.  
**Otherwise, there would be unnecessary space in the GUI**



This is the end of my first prototype for guide button and the guide itself. Next, I will create the test creation menu allowing an authorised user to create tests.

## Test creator

### Naming a test GUI

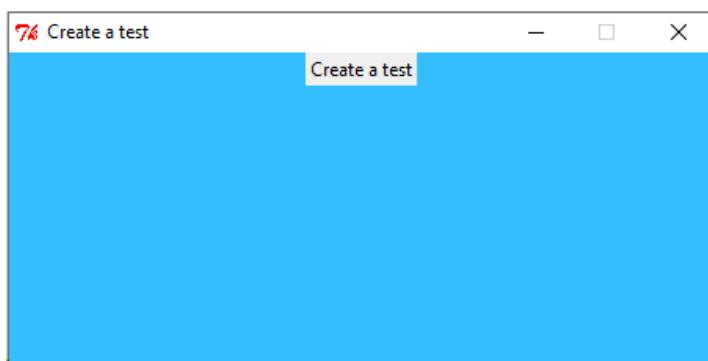
First, I create a separate window where the test will be made, and I link this to open when the “Create a test” button is pressed

```
def testCreator():
    creator=Tk()
    creator.geometry("450x200")
    creator.title("Create a test")
    creator.configure(bg="#33BFFF")

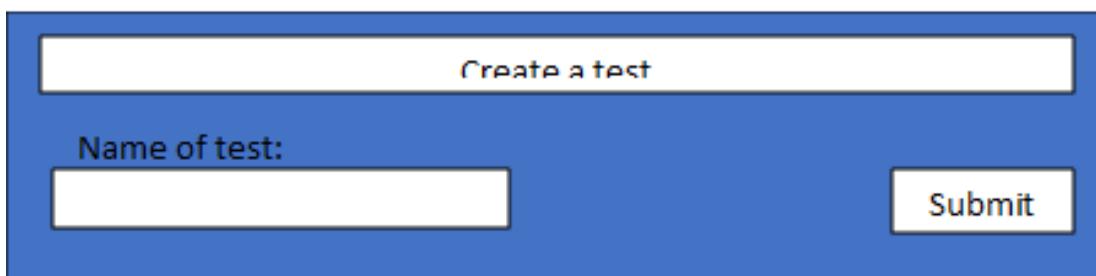
    creator.resizable(False,False)

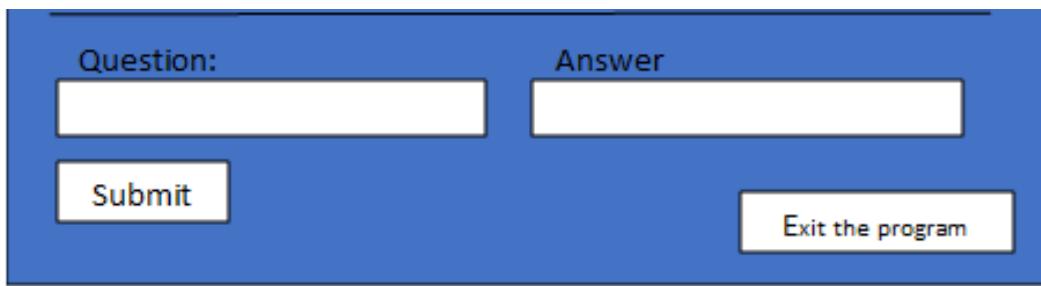
    labelCreator=Label(creator,text="Create a test")
    labelCreator.pack(side=TOP)

buttonCreate=Button(sec,text="Create a test",command=testCreator)
buttonCreate.place(x=10,y=150)
```



Before I create the test window, I realise I should split the test creator window into two separate functions and windows and remove the number of questions entry I was intending on creating. Like the design below. This would be a better alternative to having one window as it would only allow inputs for questions and answers to be entered, once a .csv file for the test has been created. **Otherwise, the user would have more inputs to enter, and the program would have to process more inputs making the program less efficient.**





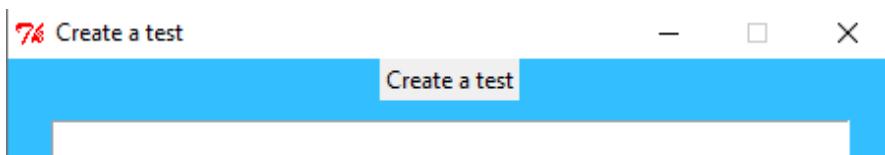
I turn the original function into the first newly suggested window, but I forget to add an appropriate label.

```
def testCreator():
    creator=Tk()
    creator.geometry("450x50")
    creator.title("Create a test")
    creator.configure(bg="#33BFFF")

    creator.resizable(False, False)

    labelCreator=Label(creator, text="Create a test")
    labelCreator.pack(side=TOP)

    nameOfTestBox=Entry(creator, width=67)
    nameOfTestBox.pack(side=BOTTOM)|
```



## Error

I add some changes to this window but receive an error.

```
def testFileCreator():
    fileCreator=Tk()
    fileCreator.geometry("450x50")
    fileCreator.title("Create a test")
    fileCreator.configure(bg="#33BFFF")

    fileCreator.resizable(False, False)

    labelCreatorTitle=Label(fileCreator, text="Create a test")
    labelCreatorTitle.pack(side=TOP)

    labelCreator=Label(fileCreator, text="Enter the name of the test file", bg="#33BFFF", font=("Courier 10 underline"), borderwidth=0)
    labelCreator.place(x=10, y=5)

    nameOfTestBox=Entry(fileCreator, width=67)
    nameOfTestBox.pack(side=BOTTOM)
```

Label added to make the input's purpose obvious to the user. **Otherwise, the user would not know what to enter into the input box**

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 176, in <lambda>
    buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(userNameBox,passwordBox))
  File "N:\GUI.py", line 193, in loginAccount
    secondaryMenu()
  File "N:\GUI.py", line 69, in secondaryMenu
    buttonCreate=Button(sec,text="Create a test",command=testCreator)
NameError: global name 'testCreator' is not defined

```

I fix this error by renaming the command for the “create a test” button along with the function, as the function name will be more appropriate for the function that creates the .csv file.

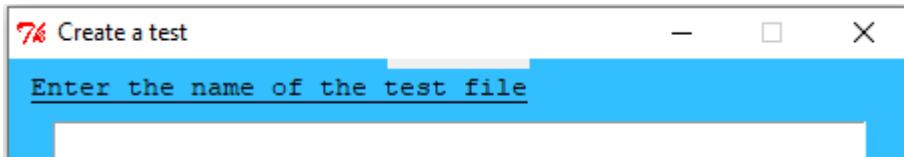
```

def nameTestFile():
    fileCreator=Tk()
    fileCreator.geometry("450x50")
    fileCreator.title("Create a test")
    fileCreator.configure(bg="#33BFFF")

    fileCreator.resizable(False,False)

buttonCreate=Button(sec,text="Create a test",command=nameTestFile)
buttonCreate.place(x=10,y=150)

```



The title does not display properly, to fix this I remove the title entirely as it is not necessary. I also reposition the label, change the label font and text and reposition the entry. This is all to make the window look slightly better.

```

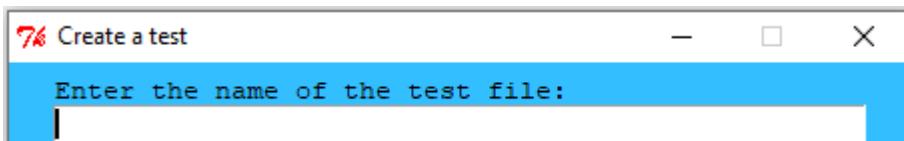
def nameTestFile():
    fileCreator=Tk()
    fileCreator.geometry("450x42")
    #fileCreator.title("Create a test")
    fileCreator.configure(bg="#33BFFF")

    fileCreator.resizable(False,False)

    labelCreator=Label(fileCreator,text="Enter the name of the test file:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
    labelCreator.place(x=22,y=5)

    nameOfTestBox=Entry(fileCreator,width=67)
    nameOfTestBox.pack(side=BOTTOM,pady=2)

```



Next, I add a ‘submit’ button to be able to execute a command using the user’s input. To place the button in a good place, I ‘place’ the entry instead of ‘packing’ it, this gives me greater control over the positioning of the elements on the window. **Otherwise, I would find it difficult to design the window.**

```

def nameTestFile():
    fileCreator=Tk()
    fileCreator.geometry("450x42")
    fileCreator.title("Create a test")
    fileCreator.configure(bg="#33BFFF")

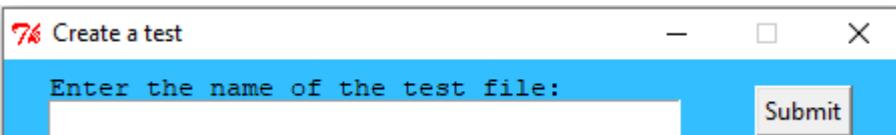
    fileCreator.resizable(False,False)

    labelCreator=Label(fileCreator,text="Enter the name of the test file:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
    labelCreator.place(x=22,y=5)

    nameOfTestBox=Entry(fileCreator,width=52)
    nameOfTestBox.place(x=22,y=20)

    submitButton=Button(fileCreator,text="Submit")
    submitButton.place(x=375,y=13)

```



## Create the test file function

Next, I create another function to create a .csv file with the inputted name. I create a temporary python file to easily test a method to create this file. I import both the os and shutil modules, the os module (which is already imported on my main program) allows me to have extra operating system operations, and the shutil module gives me access to high-level file operations. First, I find the address of the current directory, which the program is running on. I do this instead of manually writing the address as this allows the program to be run on any person's computer no matter, where the file is stored. Next, I create the destination address by concatenating a new file (newtest.csv) onto the end of the root address (this will be later replaced with the user input), then I create the source address by concatenating an empty .csv file (test.csv) onto the root address. Finally, I copy the source file into the destination, creating a new .csv file.

```

76 test.py - N:\test.py
File Edit Format Run Options Windows ↗
import os
import shutil

rootFile=os.path.abspath(os.curdir)
dstFile=rootFile+"newtest.csv"
rootFile=rootFile+"test.csv"

shutil.copy2(rootFile,dstFile)



|         |                  |                      |      |
|---------|------------------|----------------------|------|
| GUI     | 06/12/2022 15:10 | Python File          | 7 KB |
| guide   | 01/12/2022 11:51 | Text Document        | 2 KB |
| login   | 06/12/2022 14:31 | Microsoft Excel C... | 1 KB |
| test    | 06/12/2022 14:57 | Microsoft Excel C... | 1 KB |
| test    | 07/12/2022 08:59 | Python File          | 1 KB |
| newtest | 06/12/2022 14:57 | Microsoft Excel C... | 1 KB |


```

## Error

Now, I implement the solution to my actual program. I import the shutil module, then I pass the user's input through the function and link the function to execute when the 'submit' button is pressed. **Otherwise, the user cannot create the test when they are ready.**

```
from tkinter import *
import csv
import os
import shutil

def testFileCreator(testName):
    rootFile=os.path.abspath(os.curdir)
    dstFile=rootFile+testName
    rootFile=rootFile+"empty.csv"

    shutil.copy2(rootFile,dstFile)

nameOfTestBox=Entry(fileCreator,width=52)
nameOfTestBox.place(x=22,y=20)

submitButton=Button(fileCreator,text="Submit",command=testFileCreator(nameOfTestBox))
submitButton.place(x=375,y=13)
```

However, when I press the 'create a test' button' on my secondary menu, I receive an error.

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 45, in nameTestFile
    submitButton=Button(fileCreator,text="Submit",command=testFileCreator(nameOf
TestBox))
  File "N:\GUI.py", line 26, in testFileCreator
    dstFile=rootFile+testName
TypeError: Can't convert 'Entry' object to str implicitly
```

To fix this I assign the value of the entry onto a variable using the entry.get() command

```
def testFileCreator(testName):
    userTestName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    dstFile=rootFile+userTestName
    rootFile=rootFile+"empty.csv"

    shutil.copy2(rootFile,dstFile)
```

However, I get another error. To fix this I correctly name the variable.

```

def testFileCreator(testName):
    userTestName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    dstFile=rootFile+userTestName
    rootFile=rootFile+"empty.csv"

    shutil.copy2(rootFile,dstFile)

```

'usertestName' changed  
to 'userTestName'

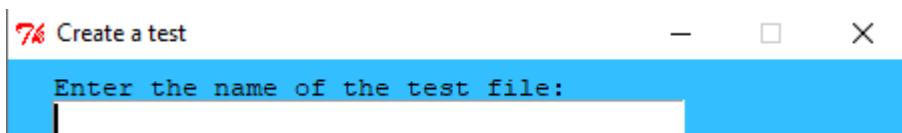
## Error

I receive another error and the submit button is not visible anymore.

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 46, in nameTestFile
    submitButton=Button(fileCreator,text="Submit",command=testFileCreator(nameOf
TextBox))
  File "N:\GUI.py", line 30, in testFileCreator
    shutil.copy2(rootFile,dstFile)
  File "C:\Python32\lib\shutil.py", line 144, in copy2
    copyfile(src, dst)
  File "C:\Python32\lib\shutil.py", line 85, in copyfile
    raise Error(`%s` and `%s` are the same file` % (src, dst))
shutil.Error: `N:\empty.csv` and `N:\empty.csv` are the same file

```



To fix this I use the lambda command to prevent the function from executing instantly. **Without this, the user does not have time to enter their test name and cannot create the test when they are ready.**

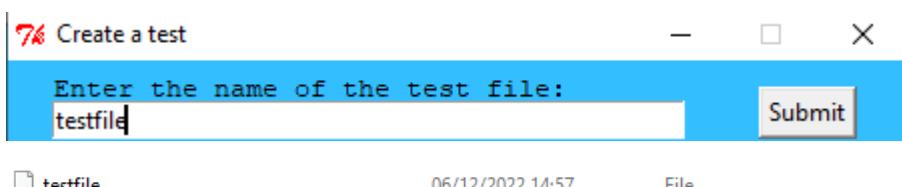
```

submitButton=Button(fileCreator,text="Submit",command=lambda: testFileCreator(nameOfTextBox))
submitButton.place(x=375,y=13)

```

## Testing

Now, it works however if .csv is not added to the end of the name the file is not made correctly. **Without this, the file cannot be accessed or modified.**

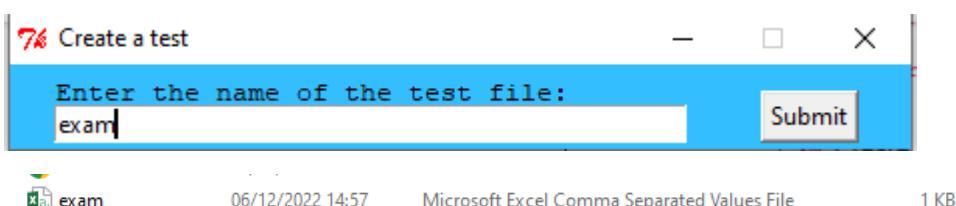


To fix this I add ".csv" to the end of the file to be created. **Without this, the file will not be in the correct format to access and modify.**

```
def testFileCreator(testName):
    userTestName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    dstFile=rootFile+userTestName+".csv"
    rootFile=rootFile+"empty.csv"

    shutil.copy2(rootFile,dstFile)
```

File extension added to create the file in the correct format.  
**Otherwise, the user would not be able to create the test using this file**



Next, I need to create a function to modify this .csv file, to allow a user to add questions and answers. I will change my previous openLogin function, which opens the 'login.csv' file, into a function that can open any .csv file. This will make the program run faster as less function need to be defined. I attempt to create this function in a copy of my python program.

```
def openLogin():
    csvFile=open('login.csv')
    csvFileContent=(csvFile)
    next(csvFile)

    global DatabaseList
    DatabaseList=[]

    for row in csvFileContent:
        database=row.strip().split(",")
        heading=['Username','Password','Admin']
        data=zip(heading,database)
        DataDict=dict(data)
        DatabaseList.append(DataDict)

    global login
    login=[]
```

```

def openCsvFile(inputtedFile):
    csvFile=open(inputtedFile)
    csvFileContent=(csvFile)
    next(csvFile)

    global DatabaseList
    DatabaseList=[]

    heading=[]
    heading=row.strip().split(",,")

    for row in csvFileContent:
        database=row.strip().split(",")
        data=zip(heading,database)
        DataDict=dict(data)
        DatabaseList.append(DataDict)

    global login
    login=[]

```

The function is given a new name for its new purpose. A parameter is introduced to the name of any .csv file to be entered. The heading list is moved before the loop as the heading is only present in the first row. The heading is also now read, instead of defined manually allowing the headings of any .csv file to be used.

## Error

However, when trying to login, an error is given.

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\test.py", line 184, in <lambda>
    buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(use
rnameBox,passwordBox))
  File "N:\test.py", line 188, in loginAccount
    openLogin()
NameError: global name 'openLogin' is not defined

```

To fix this I change the name of the function in the loginAccount function to match the new name. Changed from openLogin() to openCsvFile(login.csv). I also change the registerFunction function, which also used the previous function.

```

def loginAccount(inputtedUsername, inputtedPassword):
    openCsvFile(login.csv)
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    loginStatus=False
    global adminStatus
    adminStatus=False
    for i in DatabaseList:
        if i['Username']==username and i['Password']==password:
            messagebox.showinfo("Login successful","Login successful")
            loginStatus=True
            if i['Admin']=='testadmin':
                adminStatus=True
    if loginStatus==True:
        secondaryMenu()
    else:
        messagebox.showerror("Login attempt failed","Login attempt failed, please try again")

def registerFunction():
    openCsvFile(login.csv)
    registerWindow=Tk()
    registerWindow.geometry("450x200")
    registerWindow.title("Registration")
    registerWindow.configure(bg="#33BFFF")

    labelRegistration=Label(registerWindow, text="Registration")
    labelRegistration.pack(side=TOP)

    registerWindow.resizable(False, False)

    labelUsername=Label(registerWindow, text="Username:", bg="#33BFFF", font=("Consolas", 12))
    labelUsername.place(x=20, y=20)

    usernameBox=Entry(registerWindow, width=67)
    usernameBox.place(x=20, y=40)

    labelPassword=Label(registerWindow, text="Password:", bg="#33BFFF", font=("Consolas", 12))
    labelPassword.place(x=20, y=60)

    passwordBox=Entry(registerWindow, width=67)
    passwordBox.place(x=20, y=80)

```

## Error

I get another error, when trying to login.

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\test.py", line 184, in <lambda>
    buttonLogin=Button(loginWindow, text="Login", command=lambda :loginAccount(usernameBox,passwordBox))
  File "N:\test.py", line 188, in loginAccount
    openCsvFile(login.csv)
NameError: global name 'login' is not defined

```

To fix this I change the parameter from `login.csv` to '`login.csv`'. This is done because the program is trying to pass a variable called '`login.csv`' into the function, which is not defined in the function as a local variable. By using speech marks around the parameter, it passes the parameter as the text and not as the variable name, which does not exist. I repeat this for both times the function is used.

```
def loginAccount(inputtedUsername, inputtedPassword):
    openCsvFile('login.csv')
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    loginStatus=LoginStatus

def registerFunction():
    openCsvFile('login.csv')
    registerWindow=Tk()
    registerWindow.geometry("450x200")
    registerWindow.title("Registration")
    registerWindow.configure(bg="#33BFFF")

    labelRegistration=Label(registerWindow, text="Registration")
    labelRegistration.pack(side=TOP)

    registerWindow.resizable(False, False)
```

## Error

I get another error, when trying to login.

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\test.py", line 184, in <lambda>
    buttonLogin=Button(loginWindow, text="Login", command=lambda :loginAccount(use
rnameBox, passwordBox))
  File "N:\test.py", line 188, in loginAccount
    openCsvFile('login.csv')
  File "N:\test.py", line 87, in openCsvFile
    heading=row.strip().split(",")
UnboundLocalError: local variable 'row' referenced before assignment
```

To fix this I read the file line-by-line into a list of strings called 'row'. I then assign a variable 'row1' the first item in the list, which should be the top row (headings).

```

def openCsvFile(inputtedFile):
    csvFile=open(inputtedFile)
    csvFileContent=(csvFile)
    next(csvFile)

    row=next(csv.reader(csvFile))

    row1=row[0]

```

```

global DatabaseList
DatabaseList=[]

heading=row1.strip().split(",")

```

```

for row in csvFileContent:
    database=row.strip().split(",")
    data=zip(heading,database)
    DataDict=dict(data)
    DatabaseList.append(DataDict)

```

csv file read line by line to prevent error.

## Error

However, I get a different error.

```

>>>
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\test.py", line 187, in <lambda>
    buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(use
rnameBox,passwordBox))
  File "N:\test.py", line 198, in loginAccount
    if i['Username']==username and i['Password']==password:
KeyError: 'Username'
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\test.py", line 187, in <lambda>
    buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(use
rnameBox,passwordBox))
  File "N:\test.py", line 198, in loginAccount
    if i['Username']==username and i['Password']==password:
KeyError: 'Username'

```

I decide to create a completely new function to just open the created test file. I will try to simplify the functions into a single universal function that opens any csv file, along with this. I will also try to simplify my saveLoginData function and my future functions to save questions and answers when creating a test and save answers and score when completing a test into a single universal function later.

I revert to the program's original state

```
def openLogin():
    csvFile=open('login.csv')
    csvFileContent=(csvFile)
    next(csvFile)

    global DatabaseList
    DatabaseList=[]

    for row in csvFileContent:
        database=row.strip().split(",")
        heading=['Username','Password','Admin']
        data=zip(heading,database)
        DataDict=dict(data)
        DatabaseList.append(DataDict)

    global login
    login=[]
```

I create a function to open the test file and add the headings for the file. I link this function to execute at the end of the testFileCreator function, which creates the test file.

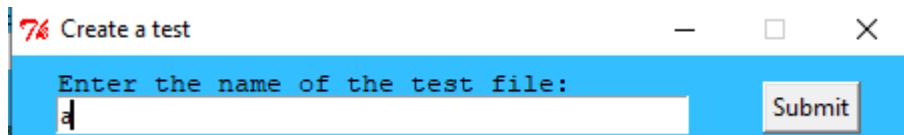
```
def openCreatedTest(testFile):
    csvFile=open(testFile,'r+')
    csvFile.write('Question,Answer\n')
    file=csvFile.read()

    for item in file:
        csvFile.write('{Question},{Answer}'.format(**item))
        csvFile.write('\n')
    csvFile.close()

def testFileCreator(testName):
    userTestName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    dstFile=rootFile+userTestName+".csv"
    rootFile=rootFile+"empty.csv"

    shutil.copy2(rootFile,dstFile)

    openCreatedTest(dstFile)
```



	A	B	C	D	E	F	G
1	Question	Answer					
2							
3							
4							
5							

## Add to created test file GUI

Next, I create a window to add questions and answers to this file. A separate function will be needed to read the input from input boxes and append them to the file each time a submit button is pressed. **Otherwise, the user cannot save the questions and answers when they are ready.**

```
def appendTestFile():
    appendWindow=Tk()
    appendWindow.geometry("450x200")
    appendWindow.title("Create a test")
    appendWindow.configure(bg="#33BFFF")

    appendWindow.resizable(False,False)

    labelTitle=Label(appendWindow,text="Add questions and answers to the test")
    labelTitle.pack(side=TOP)

    labelAddQuestion=Label(appendWindow,text="Add a question:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
    labelAddQuestion.place(x=22,y=5)

    entryAddQuestion=Entry(appendWindow)
    entryAddQuestion.place(x=22,y=20)

    labelAddAnswer=Label(appendWindow,text="Add an answer:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
    labelAddAnswer.place(x=22,y=40)

    entryAddAnswer=Entry(appendWindow)
    entryAddAnswer.place(x=22,y=55)

    submitButton=Button(appendWindow)
    submitButton.place(x=22,y=75)
```

## Testing

The program runs with no errors, but after creating a .csv file, the new window does not open. To fix this I make the window open at the end of the openCreatedTest function, which adds the headings to the file.

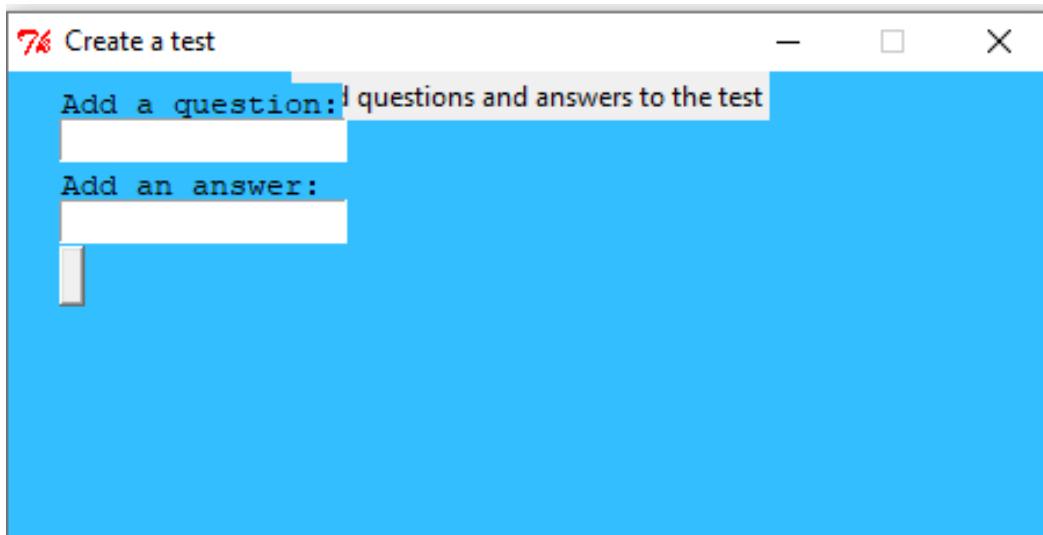
```
def openCreatedTest(testFile):
    csvFile=open(testFile,'r+')
    csvFile.write('Question,Answer\n')
    file=csvFile.read()

    for item in file:
        csvFile.write('{Question},{Answer}'.format(**item))
        csvFile.write('\n')
    csvFile.close()

appendTestFile()
```

## Testing

The program runs with no errors and the window appears; however, I need to change the coordinates of the labels and the buttons to make them fully visible. I also need to add text to the submit button to make the purpose of the button obvious to a user. **Otherwise, the user would not know what the button does.**



I make the following changes. Also, I change the entries to text boxes, this allows the user's input to be written over several lines. To fit the extra lines the window is also made larger. **Otherwise, the user will find it difficult to create long questions or answers.**

```

def appendTestFile():
    appendWindow=Tk()
    appendWindow.geometry("450x250")
    appendWindow.title("Create a test")
    appendWindow.configure(bg="#33BFFF")

    appendWindow.resizable(False,False)

    labelTitle=Label(appendWindow,text="Add questions and answers to the test")
    labelTitle.pack(side=TOP)

    labelAddQuestion=Label(appendWindow,text="Add a question:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
    labelAddQuestion.place(x=22,y=25)

    entryAddQuestion=Text(appendWindow,width=50,height=3)
    entryAddQuestion.place(x=22,y=40)

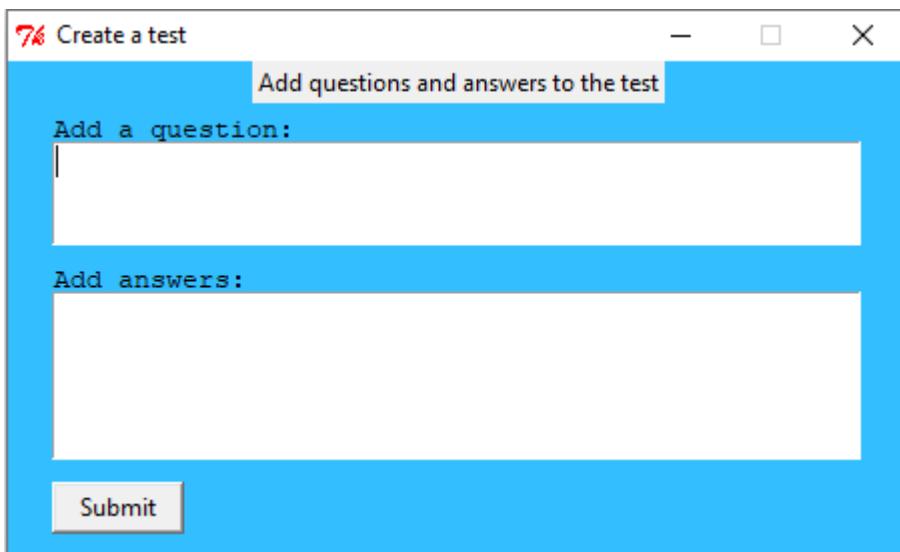
    labelAddAnswer=Label(appendWindow,text="Add answers:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
    labelAddAnswer.place(x=22,y=100)

    entryAddAnswer=Text(appendWindow,width=50,height=5)
    entryAddAnswer.place(x=22,y=115)

    submitButton=Button(appendWindow,text="Submit",width=8)
    submitButton.place(x=22,y=210)

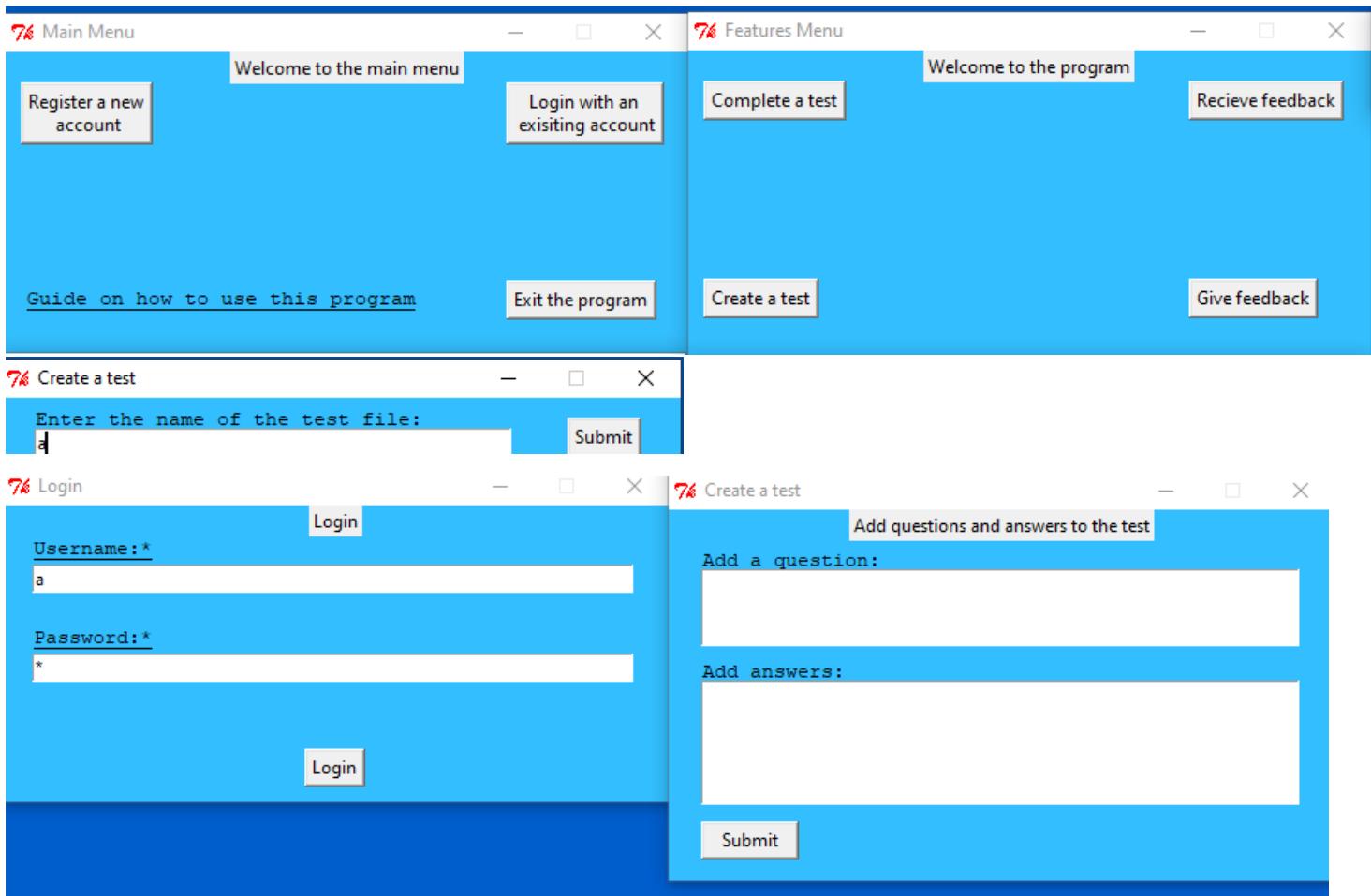
```

GUI elements resized and moved. **Otherwise, the user will find it difficult to create long questions or answers.**



### Closing windows after they have been used

During the execution of the program, several windows are used to access parts of the program. Some of these windows are not needed after they are used, for example: the login window. Therefore, I will add a command to close unnecessary windows, once they are used. **Otherwise, unnecessary windows will stay open, this hinders the performance of the program.**



## Error

First, I try the `destroy()` command, which should close the window. However, I receive an error.

```
def loginAccount(inputtedUsername, inputtedPassword):
    openLogin()
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    loginStatus=False
    global adminStatus
    adminStatus=False
    for i in DatabaseList:
        if i['Username']==username and i['Password']==password:
            messagebox.showinfo("Login successful","Login successful")
            loginStatus=True
            loginWindow.destroy()
            if i['Admin']=='testadmin':
                adminStatus=True
    if loginStatus==True:
        secondaryMenu()
    else:
        messagebox.showerror("Login attempt failed","Login attempt failed, please try again")
```

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 221, in <lambda>
    buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(usernameBox,passwordBox))
  File "N:\GUI.py", line 235, in loginAccount
    loginWindow.destroy()
NameError: global name 'loginWindow' is not defined
```

To fix this I pass the name 'loginWindow' into the function. **Otherwise, I would receive an error message.**

```
buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(usernameBox,passwordBox,loginWindow))
buttonLogin.pack(side=BOTTOM, pady=10)
```

```
def loginAccount(inputtedUsername,inputtedPassword,loginWindow):
    #code here
```

I repeat the following for the window where the usernames their file. **Otherwise, I would receive an error message.**

```
submitButton=Button(fileCreator,text="Submit",command=lambda: testFileCreator(nameOfTestBox,fileCreator))
submitButton.place(x=375,y=13)
```

```
def testFileCreator(testName,fileCreator):
    userTestName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    dstFile=rootFile+userTestName+".csv"
    rootFile=rootFile+"empty.csv"

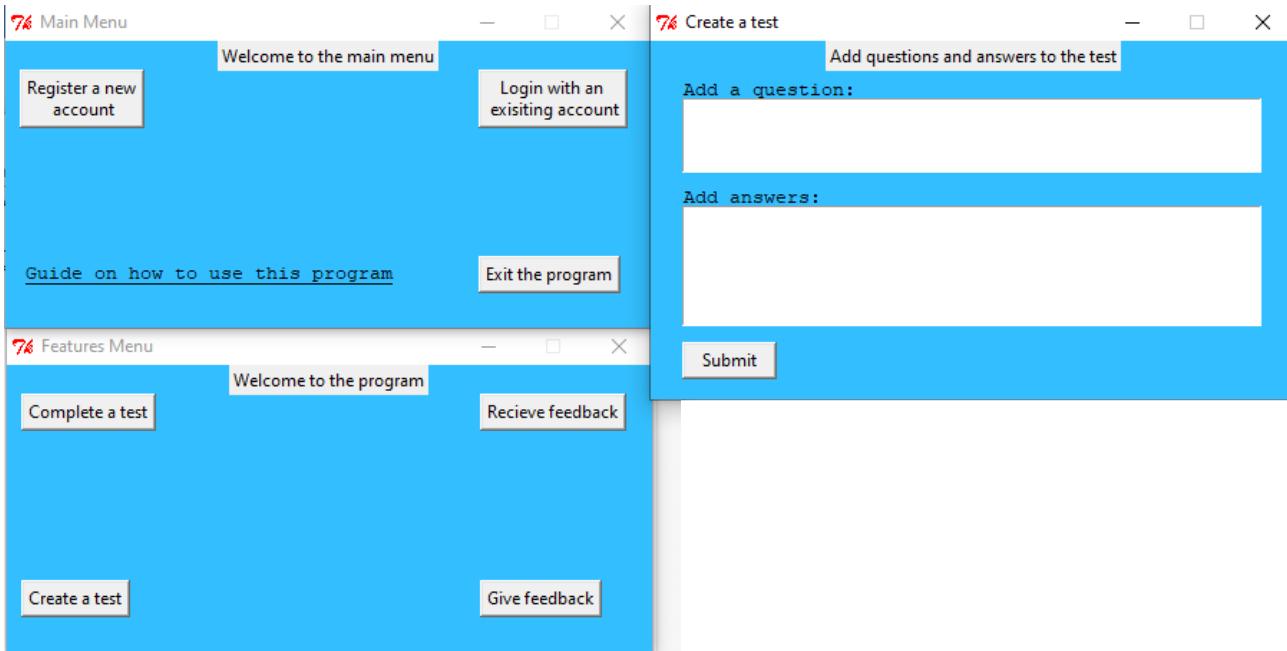
    shutil.copy2(rootFile,dstFile)

    fileCreator.destroy()

    openCreatedTest(dstFile)
```

## Testing

The program runs successfully and only the necessary windows are kept open. **Without this, unnecessary windows will stay open, this hinders the performance of the program.**



## Removing all global variables

Next, I want to remove all global variables, **otherwise they may be modified accidentally at a later stage in the program causing errors**. Instead, I will pass the local variables into the functions when needed.

I start with the openLogin function.

```
def openLogin():
    csvFile=open('login.csv')
    csvFileContent=(csvFile)
    next(csvFile)

    global DatabaseList
    DatabaseList=[]

    for row in csvFileContent:
        database=row.strip().split(",")
        heading=['Username','Password','Admin']
        data=zip(heading,database)
        DataDict=dict(data)
        DatabaseList.append(DataDict)

    global login
    login=[]
```

Global variables removed. Otherwise, the variables could be modified at a later stage in the program and could cause errors

```

def openLogin():
    csvFile=open('login.csv')
    csvFileContent=(csvFile)
    next(csvFile)

    DatabaseList=[]

    for row in csvFileContent:
        database=row.strip().split(",")
        heading=['Username','Password','Admin']
        data=zip(heading,database)
        DataDict=dict(data)
        DatabaseList.append(DataDict)

    login=[]

```

Then, I make this function execute either the register or login function, this will allow the necessary array to be passed through the functions. **Otherwise, the necessary array cannot be used the following function.**

```

def openLogin(buttonOutput):
    csvFile=open('login.csv')
    csvFileContent=(csvFile)
    next(csvFile)

    DatabaseList=[]

    for row in csvFileContent:
        database=row.strip().split(",")
        heading=['Username','Password','Admin']
        data=zip(heading,database)
        DataDict=dict(data)
        DatabaseList.append(DataDict)

    login=[]

    if buttonOutput==startLogin:
        loginFunction()
    elif buttonOutput==startRegister:
        registerFunction()

```

```

buttonRegistration=Button(root,text="Register a new\n account",command=openLogin(startRegister))
buttonRegistration.place(x=10, y=20)

buttonLogin=Button(root,text="Login with an\n exisiting account",command=openLogin(startLogin))
buttonLogin.place(x=330,y=20)

```

## Error

When the program is run a name error is given.

```

Traceback (most recent call last):
  File "N:\GUI.py", line 280, in <module>
    buttonRegistration=Button(root,text="Register a new\n account",command=open
Login(startRegister))
NameError: name 'startRegister' is not defined

```

To try and fix this I turn the arguments into text using speech marks.

```

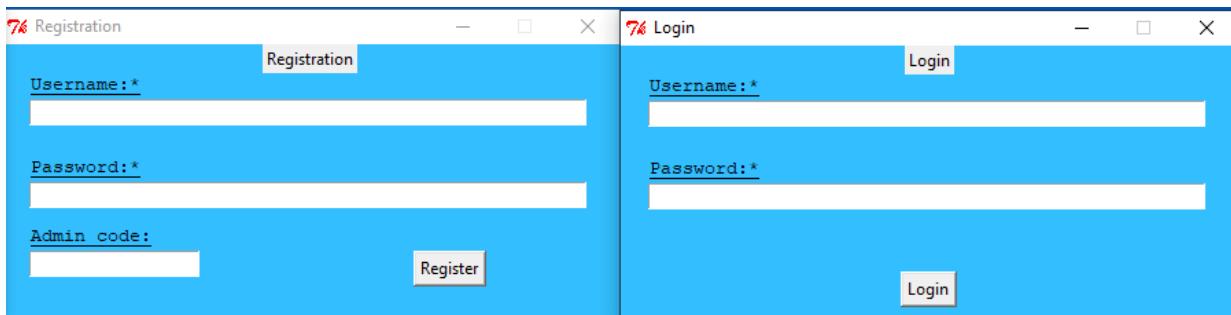
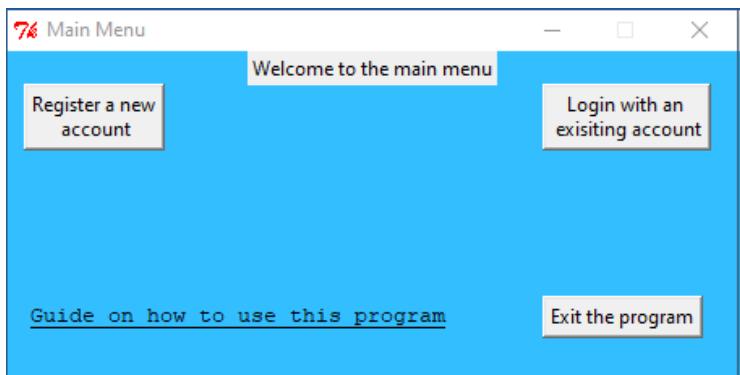
buttonRegistration=Button(root,text="Register a new\n account",command=openLogin("startRegister"))
buttonRegistration.place(x=10, y=20)

buttonLogin=Button(root,text="Login with an\n exisiting account",command=openLogin("startLogin"))
buttonLogin.place(x=330,y=20)

if buttonOutput=="startLogin":
    loginFunction()
elif buttonOutput=="startRegister":
    registerFunction()

```

However, when the program is run both functions are executed and therefore both windows open.



I fix this by adding the lambda function. **Otherwise, both functions will be executed as soon as they are defined.** This stops the functions being read when the program is executed, this prevents the windows from opening as all windows will open when being defined within a function. **Otherwise, windows will be opened without the user selecting the respective option.**

```

buttonRegistration=Button(root,text="Register a new\n account",command=lambda: openLogin("startRegister"))
buttonRegistration.place(x=10, y=20)

buttonLogin=Button(root,text="Login with an\n exisiting account",command=lambda: openLogin("startLogin"))
buttonLogin.place(x=330,y=20)

```

## Error

However, an error is given both when the registration menu opens and when the login menu opens.

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 229, in <lambda>
    buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(usernameBox,passwordBox,loginWindow))
  File "N:\GUI.py", line 238, in loginAccount
    for i in DatabaseList:
NameError: global name 'DatabaseList' is not defined
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 199, in <lambda>
    buttonRegister=Button(registerWindow,text="Register",command=lambda :registerAccount(usernameBox,passwordBox,adminBox))
  File "N:\GUI.py", line 148, in registerAccount
    openLogin()
```

To fix this, I first pass the 'login' array into all register functions. However, instead of creating the array in the openLogin function, I now create it in the registerAccount function, this optimises the program by reducing the overall arguments and parameters. **Without this, the program would take longer to execute these functions.**

```
def registerAccount(inputtedUsername,inputtedPassword,inputtedAdmin):
    login=[]
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    admin=inputtedAdmin.get()
    logindata={'Username':username,'Password':password,'Admin':admin}
    login.append(logindata)
    saveLoginData(login)

def saveLoginData(login):
    -----
```

Then I pass the 'databaseList' array into all login functions. **Otherwise, this array cannot be used by the other login functions.** I also change the name of the array from DatabaseList to databaseList. **Otherwise, the array name will not match the format of all other variables, which may make me reference the array by an incorrect name in the future.**

```
if buttonOutput=="startLogin":
    loginFunction(databaseList)

def loginFunction(databaseList):

buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(usernameBox,passwordBox,loginWindow,databaseList))
buttonLogin.pack(side=BOTTOM, pady=10)

def loginAccount(inputtedUsername,inputtedPassword,loginWindow,databaseList):
    ... ...
```

Next, I remove the global adminStatus variable. **Otherwise, this variable could be modified later in the program, which may cause an error.**

```
def loginAccount(inputtedUsername, inputtedPassword, loginWindow, databaseList):
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    loginStatus=False
    global adminStatus
    adminStatus=False
    . . .
```

I remove the global adminStatus variable.  
**Otherwise, this variable could be modified later in the program, which may cause an error.**

Then, I pass the variable into the secondary menu. This is the only place where the variable is used. **Otherwise, the variable would become unusable as it is no longer a global variable it must be passed into the program to be accessed.**

```
for i in databaseList:
    if i['Username']==username and i['Password']==password:
        messagebox.showinfo("Login successful","Login successful")
        loginStatus=True
        loginWindow.destroy()
        if i['Admin']=='testadmin':
            adminStatus=True
    if loginStatus==True:
        secondaryMenu(adminStatus)
    else:
        messagebox.showerror("Login attempt failed","Login attempt failed, please try again")

def secondaryMenu(adminStatus):
    sec=Tk()
    sec.geometry("450x200")
    sec.title("Features Menu")
    sec.configure(bg="#33BFFF")
```

Now, all global variables have been removed.

### Retrieve questions and answer/answers and save them to the test file

Next, I create a function to get the inputted questions and answers from the text boxes. **Otherwise, the user's inputs cannot be saved into the test file**. This is essentially the same as my registerAccount function. Later I will try to create one function for both. The reason I can't use the registerAccount function to save the questions and answers is because both functions need a different number of parameters, and both have manually defined dictionaries.

```
def appendDetails(inputtedQuestion, inputtedAnswers):
    csv=[]
    question=inputtedQuestion.get()
    answers=inputtedAnswers.get()
    csvdata={'Question':username, 'Answer'}
    csv.append(csvData)

submitButton=Button	appendWindow, text="Submit", width=8, command=lambda: appendDetails(entryAddQuestion, entryAddAnswer))
submitButton.place(x=22, y=210)
```

Next, I create a function to save the details to the .csv file. **Otherwise, the user's inputs cannot be saved into the test file.** This is essentially the same as my saveLoginData function. Later I will try to create one function for both.

When creating this function, I realise I need to pass the name of the test file through several functions, so I define parameters and add arguments. **Otherwise, the user's inputs cannot be saved into the test file as the test file's name is required to open the correct .csv file.**

```

def openCreatedTest(testFile):
    csvFile=open(testFile,'r+')
    csvFile.write('Question,Answer\n')
    file=csvFile.read()

    for item in file:
        csvFile.write('{Question},{Answer}'.format(**item))
        csvFile.write('\n')
    csvFile.close()

    appendTestFile(testFile)

def appendTestFile(testFile):
    appendWindow=Tk()
    appendWindow.geometry("450x250")
    appendWindow.title("Create a test")
    appendWindow.configure(bg="#33BFFF")

    appendWindow.resizable(False,False)

    labelTitle=Label(appendWindow,text="Add questions and answers to the test")
    labelTitle.pack(side=TOP)

    labelAddQuestion=Label(appendWindow,text="Add a question:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
    labelAddQuestion.place(x=22,y=25)

    entryAddQuestion=Text(appendWindow,width=50,height=3)
    entryAddQuestion.place(x=22,y=40)

    labelAddAnswer=Label(appendWindow,text="Add answers:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
    labelAddAnswer.place(x=22,y=100)

    entryAddAnswer=Text(appendWindow,width=50,height=5)
    entryAddAnswer.place(x=22,y=115)

    submitButton=Button(appendWindow,text="Submit",width=8,command=lambda: appendDetails(entryAddQuestion,entryAddAnswer,testFile))
    submitButton.place(x=22,y=210)

def appendDetails(inputtedQuestion,inputtedAnswers,testFile):
    csv=[]
    question=inputtedQuestion.get()
    answers=inputtedAnswers.get()
    csvdata={'Question':question,'Answer':answers}
    csv.append(csvdata,testFile)

```

I then create the function and link it to execute after the questions and answers have been read. **Otherwise, the user's inputs cannot be saved once they have been acquired.**

```

def saveTestData(csv,file):
    try:
        fileHandle=open(file,'r+')
        fileContent=fileHandle.read()
        if fileContent.strip()=='':
            fileHandle.write('Question,Answer\n')
        for item in csv:
            fileHandle.write('{Question},{Answer}'.format(**item))
            fileHandle.write('\n')
        fileHandle.close()
    except OSError:
        print('Can\'t write to file')

def appendDetails(inputtedQuestion,inputtedAnswers,testFile):
    csv=[]
    question=inputtedQuestion.get()
    answers=inputtedAnswers.get()
    csvData={'Question':username,'Answer'}
    csv.append(csvData,testFile)

    saveTestData(csv,testFile)

```

When the program is run, I get a syntax error.

```

def saveTestData(csv,file):
    try:
        fileHandle=open(file,'r+')
        fileContent=fileHandle.read()
        if fileContent.strip()= '':
            fileHandle.write('Question,Answer\n')
        for item in csv:
            fileHandle.write('{Question},{Answer}'.format(**item))
            fileHandle.write('\n')
        fileHandle.close()
    except OSError:
        print('Can\'t write to file')

```

I fix this error by changing the = to ==. Otherwise, the program cannot check if the csv file has no headings in order to determine whether or not to add headings or not, which is needed to format the data added.

```

-----
if fileContent.strip()== '':
    fileHandle.write('Question,Answer\n')

```

I get another syntax error, when the program is run.

```

def appendDetails(inputtedQuestion,inputtedAnswers,testFile):
    csv=[]
    question=inputtedQuestion.get()
    answers=inputtedAnswers.get()
    csvData={'Question':username,'Answer'}
    csv.append(csvData,testFile)

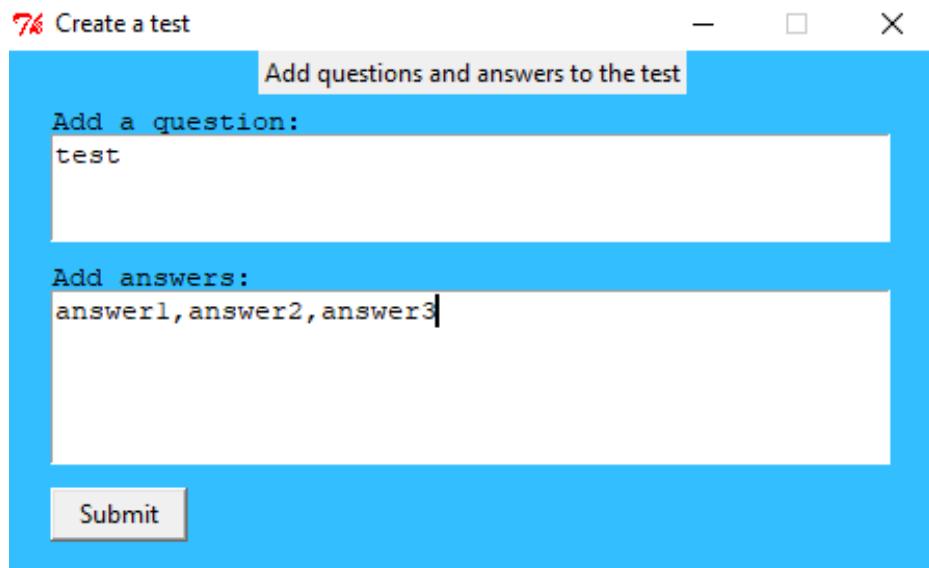
```

I fix this by completing the dictionary. Otherwise, the inputted data is not formatted to be added to the database.

```
csvdata={ 'Question':username, 'Answer':answers}
csv.append(csvData,testFile)
```

## Error

The program runs, however when the submit button is pressed, an error is given.



```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 70, in <lambda>
    submitButton=Button	appendWindow, text="Submit", width=8, command=lambda: appendDetails(entryAddQuestion, entryAddAnswer, testFile))
  File "N:\GUI.py", line 40, in appendDetails
    question=inputtedQuestion.get()
TypeError: get() takes at least 2 arguments (1 given)
```

## Error

To fix this I need to change my arguments for the get() method to get(startindex [,endindex]), this is because the get() method has different parameters depending on whether a text or entry widget is used.

After adding new parameter, I get a new error.

```
question=inputtedQuestion.get(0,END)
answers=inputtedAnswers.get(0,END)
```

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 71, in <lambda>
    submitButton=Button	appendWindow,text="Submit",width=8,command=lambda: appendDetails(entryAddQuestion,entryAddAnswer,testFile))
  File "N:\GUI.py", line 41, in appendDetails
    question=inputtedQuestion.get(0,END)
  File "C:\Python32\lib\tkinter\__init__.py", line 2944, in get
    return self.tk.call(self._w, 'get', index1, index2)
_tkinter.TclError: bad text index "0"

```

## Error

I try to fix this by changing the start index to 1, but this also gives an error.

```

question=inputtedQuestion.get(1,END)
answers=inputtedAnswers.get(1,END)

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 71, in <lambda>
    submitButton=Button	appendWindow,text="Submit",width=8,command=lambda: appendDetails(entryAddQuestion,entryAddAnswer,testFile))
  File "N:\GUI.py", line 41, in appendDetails
    question=inputtedQuestion.get(1,END)
  File "C:\Python32\lib\tkinter\__init__.py", line 2944, in get
    return self.tk.call(self._w, 'get', index1, index2)
_tkinter.TclError: bad text index "1"

```

## Error

I then fix this by setting the start index to 1.0. However, a name error.

```

question=inputtedQuestion.get(1.0,END)
answers=inputtedAnswers.get(1.0,END)

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 71, in <lambda>
    submitButton=Button	appendWindow,text="Submit",width=8,command=lambda: appendDetails(entryAddQuestion,entryAddAnswer,testFile))
  File "N:\GUI.py", line 43, in appendDetails
    csvdata={'Question':username,'Answer':answers}
NameError: global name 'username' is not defined
'
```

I fix this by correcting the value of the dictionary.

```
:csvdata={'Question':question, 'Answer':answers}
```

## Error

When the program is run, I get another name error.

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 71, in <lambda>
    submitButton=Button	appendWindow, text="Submit", width=8, command=lambda: appendDetails(entryAddQuestion, entryAddAnswer, testFile))
  File "N:\GUI.py", line 44, in appendDetails
    csv.append(csvData, testFile)
NameError: global name 'csvData' is not defined
```

I fix this by correcting the name of the dictionary from csvdata to csvData.

```
csvData={'Question':question, 'Answer':answers}
csv.append(csvData, testFile)
```

## Error

When the program is run, I get a type error.

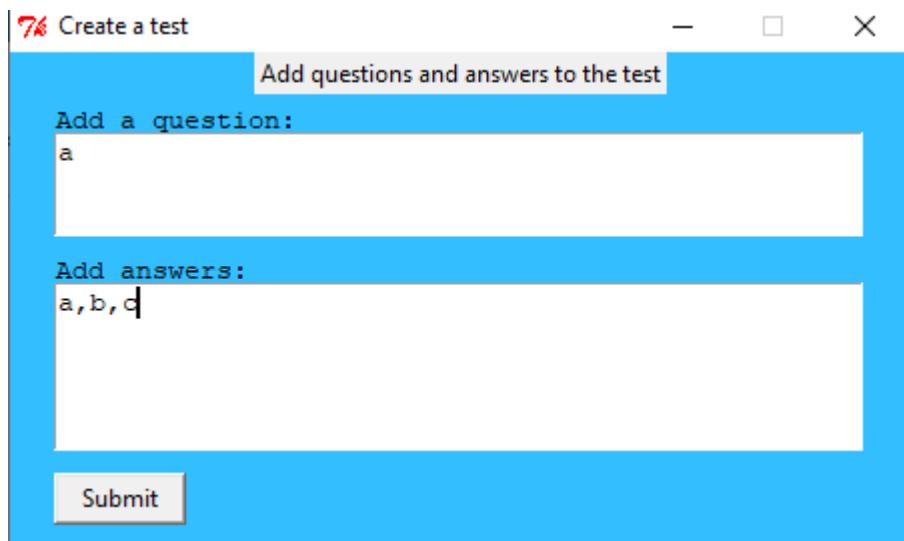
```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 71, in <lambda>
    submitButton=Button	appendWindow, text="Submit", width=8, command=lambda: appendDetails(entryAddQuestion, entryAddAnswer, testFile))
  File "N:\GUI.py", line 44, in appendDetails
    csv.append(csvData, testFile)
TypeError: append() takes exactly one argument (2 given)
```

To fix this, I remove testFile from the arguments of csv.append().

```
csv.append(csvData)
```

## Testing

When the program is run the details are added, however it is not in the correct format.

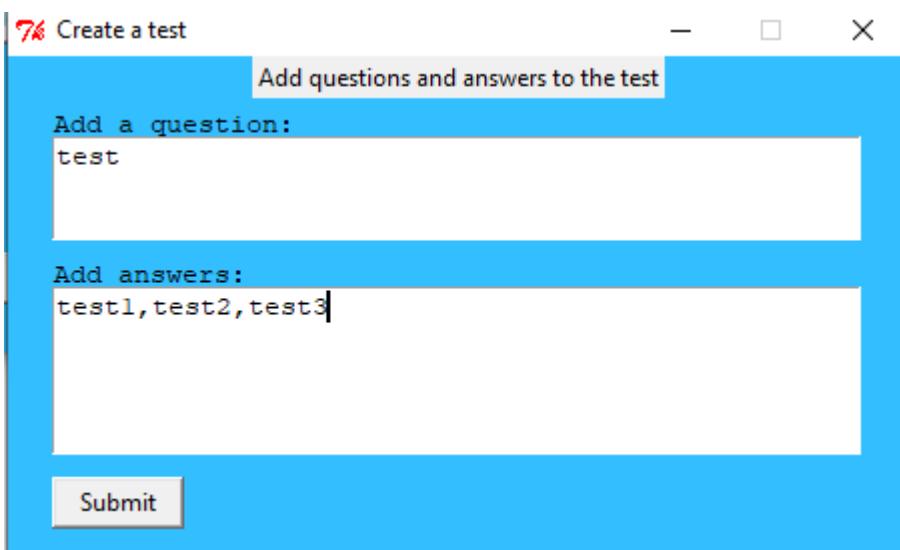


Question	Answer		
a			
	a	b	c

I learn that tkinter automatically adds a trailing newline to the end all text widgets, this causes the answers to appear on a new line in the csv file. To fix this I modify the last parameter of the get() function that makes the function get all values within the text widget minus the last character, which is the trailing newline, this prevents a new line from being made. **Without this, the answers would not be in the correct format.**

```
def appendDetails(inputtedQuestion, inputtedAnswers, testFile):
    csv=[]
    question=inputtedQuestion.get(1.0, 'end-1c')
    answers=inputtedAnswers.get(1.0, 'end-1c')
    csvData={'Question':question, 'Answer':answers}
    csv.append(csvData)

    saveTestData(csv, testFile, csvData)
```



	A	B	C	D
1	Question	Answer		
2	test	test1	test2	test3
3				

I add an if statement to execute the function if both the question and answers text boxes are filled. **Otherwise, if one or both boxes are not filled, the test will not display correctly when being completed, either questions or answers will not appear, or both will not appear.**

```

def appendDetails(inputtedQuestion, inputtedAnswers, testFile):
    csv=[]
    question=inputtedQuestion.get(1.0, 'end-1c')
    answers=inputtedAnswers.get(1.0, 'end-1c')
    csvData={'Question':question, 'Answer':answers}
    csv.append(csvData)

    if len(question)!=0 and len(answers)!=0:
        saveTestData(csv, testFile, csvData)

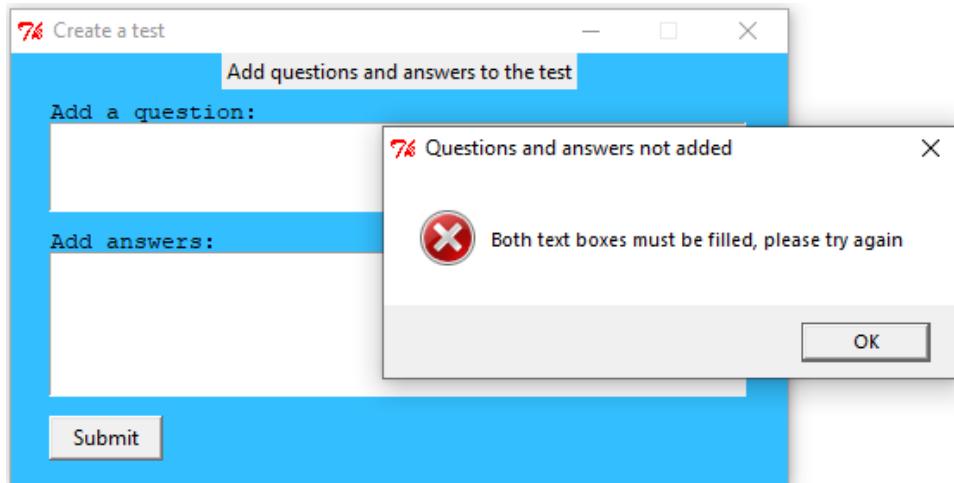
```

I add a message box that appears if the if statement is not met. **Otherwise, there is no indicator to the user if the questions and answers failed to submit.**

```

if len(question)!=0 and len(answers)!=0:
    saveTestData(csv, testFile, csvData)
else:
    messagebox.showerror("Questions and answers not added", "Both text boxes must be filled, please try again")

```



I also add a message box that appears if the questions and answers are successfully added. **Otherwise, there is no indicator to the user if the questions and answers are saved successfully.**

```

def saveTestData(csv, file, csvData):
    try:
        fileHandle=open(file, 'r+')
        fileContent=fileHandle.read()
        if fileContent.strip()=='':
            fileHandle.write('Question,Answer\n')
        for item in csv:
            fileHandle.write('{Question},{Answer}'.format(**csvData))
            fileHandle.write('\n')
        messagebox.showinfo("Success", "Question and answers added successfully")

        fileHandle.close()
    except OSError:
        print('Can\'t write to file')

```

## Completing the test creation GUI

Next, I create a button to finish creating the test. **Otherwise, the user cannot end the test creation process.**

```

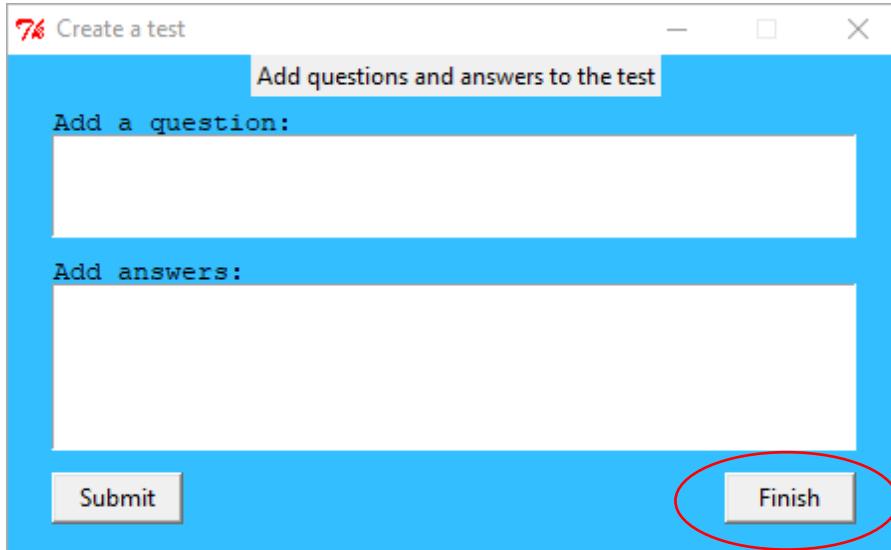
entryAddAnswer=Text(appendWindow,width=50,height=5)
entryAddAnswer.place(x=22,y=115)

submitButton=Button(appendWindow,text="Submit",width=8,command=lambda: appendDetails(entryAddQuestion,entryAddAnswer,testFile))
submitButton.place(x=22,y=210)

finishButton=Button(appendWindow,text="Finish",width=8)
finishButton.place(x=360,y=210)

```

I create a button to finish creating the test. Otherwise, the user cannot end the test creation process.



Next, I create a function for the finish button. This will open a new window that will allow the user to preview the test they have made. I link this function to the finish button.

```

def testCreatorEnd(file):
    previewWindow=Tk()
    previewWindow.geometry("450x250")
    previewWindow.title("Test preview")
    previewWindow.configure(bg="#33BFFF")

    test=open(file,'r+')
    testData=test.read()

    scrollbar=Scrollbar(previewWindow)
    scrollbar.pack(side=RIGHT,fill=Y)

    testPreview=Text(previewWindow,bg="#33BFFF",WIDTH=450,wrap=WORD)
    testPreview.pack()

    testPreview.insert(END,testData)

    testPreview.config(yscrollcommand=scrollbar.set,state=DISABLED)
    scrollbar.config(command=testPreview.yview)

    finishButton=Button(appendWindow,text="Finish",width=8,command=lambda: testCreatorEnd(testFile))
    finishButton.place(x=360,y=210)

```

## Error

However, when the program is run, I get an error as soon as I press the finish button.

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 101, in <lambda>
    finishButton=Button	appendWindow, text="Finish", width=8, command=lambda: testCreatorEnd(testFile)
  File "N:\GUI.py", line 36, in testCreatorEnd
    testPreview=Text(previewWindow, bg="#33BFFF", WIDTH=450, wrap=WORD)
  File "C:\Python32\lib\tkinter\__init__.py", line 2811, in __init__
    Widget.__init__(self, master, 'text', cnf, kw)
  File "C:\Python32\lib\tkinter\__init__.py", line 1958, in __init__
    (widgetName, self._w) + extra + self._options(cnf))
_tkinter.TclError: unknown option "-WIDTH"

```

I fix this by changing the parameter from WIDTH to width.

```

testPreview=Text(previewWindow, bg="#33BFFF", width=450, wrap=WORD)
testPreview.pack()

```



## Preview test button

I decide to change the purpose of this function. Instead, the button previewing the test then closing the test creator menu. I will make this button now say 'Preview' and it can be pressed at any time while making the test. **Otherwise, the user cannot view the test during its creation.** Then I make a new function called 'Finish', which simply closes the window. I also rearrange the positions of the buttons and give the testCreatorEnd function a better name for its new purpose.

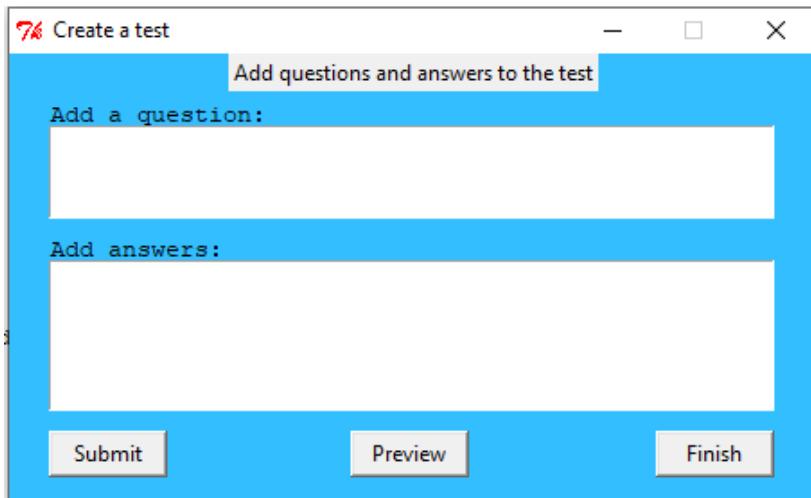
```

previewButton=Button	appendWindow, text="Preview", width=8, command=lambda: testPreview(testFile)
previewButton.place(x=190,y=210)

finishButton=Button	appendWindow, text="Finish", width=8, command=lambda: appendWindow.destroy()
finishButton.place(x=360,y=210)

def testPreview(file):

```



## Finish test creation button / Grade boundaries GUI

I also decide to make a separate function for the 'Finish' button as another window will need to be made to input grade boundaries.

```
finishButton=Button	appendWindow, text="Finish", width=8, command=lambda: endTest())
finishButton.place(x=360, y=210)

def endTest():
    #grade boundaries code

    appendWindow.destroy()
```

## Error

However, when the button is pressed a name error is received

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 110, in <lambda>
    finishButton=Button.appendWindow, text="Finish", width=8, command=lambda: endTe
st())
  File "N:\GUI.py", line 26, in endTest
    appendWindow.destroy()
NameError: global name 'appendWindow' is not defined
```

To fix this I add the appendWindow window as a parameter to the function

```
finishButton=Button.appendWindow, text="Finish", width=8, command=lambda: endTest(appendWindow))
finishButton.place(x=360, y=210)

def endTest(closeWindow):
    closeWindow.destroy()
```

Next, I will create a window to display the number of marks in the newly created test file and give several input boxes to input grade boundaries. **Otherwise, the user cannot set grade boundaries for a test.**

First, I create the window and place all the input boxes

## Grade boundaries GUI

```
def endTest(closeWindow):
    gradeWindow=Tk()
    gradeWindow.geometry("300x150")
    gradeWindow.title("Add grade boundaries")
    gradeWindow.configure(bg="#33BFFF")

    gradeTitle=Label(gradeWindow,text="Add grade boundaries to the test")
    gradeTitle.pack(side=TOP)

    entryAgrade=Text(gradeWindow,width=5,height=1)
    entryAgrade.place(x=20,y=40)

    entryBgrade=Text(gradeWindow,width=5,height=1)
    entryBgrade.place(x=125,y=40)

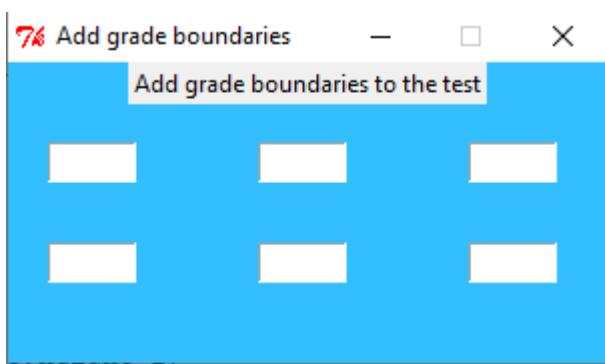
    entryCgrade=Text(gradeWindow,width=5,height=1)
    entryCgrade.place(x=230,y=40)

    entryDgrade=Text(gradeWindow,width=5,height=1)
    entryDgrade.place(x=20,y=90)

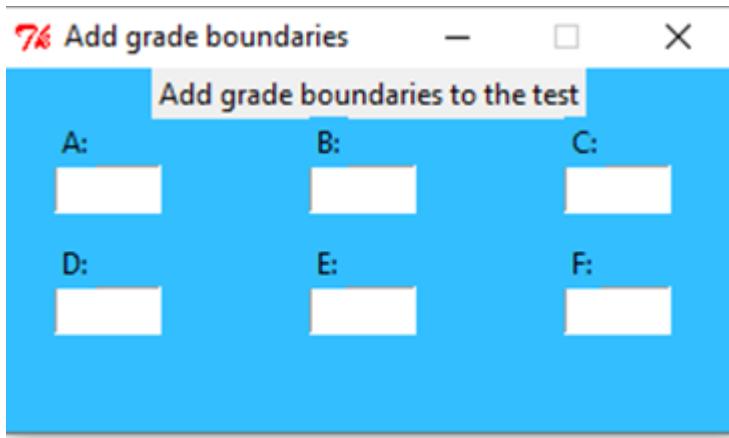
    entryEgrade=Text(gradeWindow,width=5,height=1)
    entryEgrade.place(x=125,y=90)

    entryFgrade=Text(gradeWindow,width=5,height=1)
    entryFgrade.place(x=230,y=90)

    gradeWindow.resizable(False,False)
```



Next, I add labels to all the input boxes. Otherwise, the user does not know what to input into the input boxes.



```
gradeAlabel=Label(gradeWindow,bg="#33BFFF",text="A:")
gradeAlabel.place(x=20,y=20)

gradeBlabel=Label(gradeWindow,bg="#33BFFF",text="B:")
gradeBlabel.place(x=125,y=20)

gradeClabel=Label(gradeWindow,bg="#33BFFF",text="C:")
gradeClabel.place(x=230,y=20)

gradeDlabel=Label(gradeWindow,bg="#33BFFF",text="D:")
gradeDlabel.place(x=20,y=70)

gradeElabel=Label(gradeWindow,bg="#33BFFF",text="E:")
gradeElabel.place(x=125,y=70)

gradeFlabel=Label(gradeWindow,bg="#33BFFF",text="F:")
gradeFlabel.place(x=230,y=70)
```

I realise the 'B' and 'C' labels slightly overlap into the title label, so I slightly move all labels and boxes down

```

entryAgrade=Text(gradeWindow,width=5,height=1)
entryAgrade.place(x=20,y=45)

entryBgrade=Text(gradeWindow,width=5,height=1)
entryBgrade.place(x=125,y=45)

entryCgrade=Text(gradeWindow,width=5,height=1)
entryCgrade.place(x=230,y=45)

entryDgrade=Text(gradeWindow,width=5,height=1)
entryDgrade.place(x=20,y=95)

entryEgrade=Text(gradeWindow,width=5,height=1)
entryEgrade.place(x=125,y=95)

entryFgrade=Text(gradeWindow,width=5,height=1)
entryFgrade.place(x=230,y=95)

gradeAlabel=Label(gradeWindow,bg="#33BFFF",text="A:")
gradeAlabel.place(x=20,y=25)

gradeBlabel=Label(gradeWindow,bg="#33BFFF",text="B:")
gradeBlabel.place(x=125,y=25)

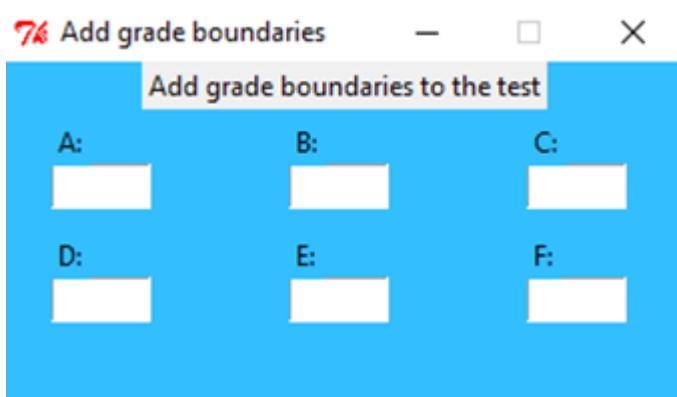
gradeClabel=Label(gradeWindow,bg="#33BFFF",text="C:")
gradeClabel.place(x=230,y=25)

gradeDlabel=Label(gradeWindow,bg="#33BFFF",text="D:")
gradeDlabel.place(x=20,y=75)

gradeElabel=Label(gradeWindow,bg="#33BFFF",text="E:")
gradeElabel.place(x=125,y=75)

gradeFlabel=Label(gradeWindow,bg="#33BFFF",text="F:")
gradeFlabel.place(x=230,y=75)

```



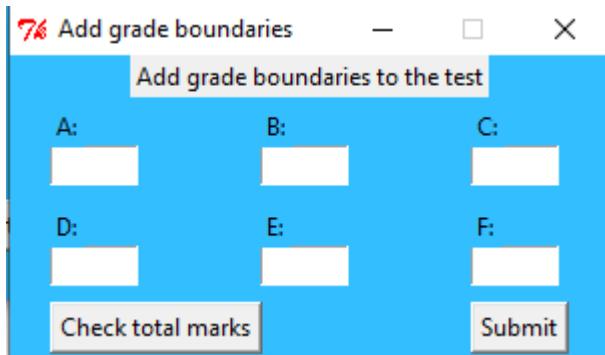
Next, I add a button to check the total marks of the test, and a button to input the grade boundaries. **Otherwise, the user cannot check the total marks in a test, which will influence the grade boundaries.**

```

checkButton=Button(gradeWindow,text="Check total marks")
checkButton.place(x=20,y=123)

submitButton=Button(gradeWindow,text="Submit")
submitButton.place(x=230,y=123)

```



## Check total marks button

Next, I create the check total marks function. This function will read how many lines there are in the rows there are in the csv file and will use this to add up the total marks, with one row being one mark minus the first row, which contains the question and answer headings.

```
def checkMarks(file):
    marks=-1
    content=open(file,'r+')
    fileContent=content.read()

    for item in fileContent:
        marks=marks+1

    messagebox.showinfo("Total marks",marks)
```

I also pass the file through several functions so the check total marks function can read the csv file

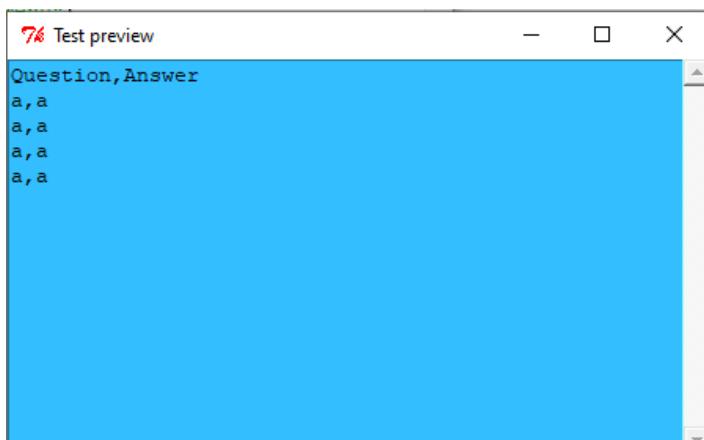
```
finishButton=Button(appendWindow,text="Finish",width=8,command=lambda: endTest(appendWindow,testFile))
finishButton.place(x=360,y=210)

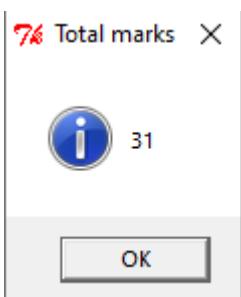
def endTest(closeWindow,file):

    checkButton=Button(gradeWindow,text="Check total marks",command=lambda: checkMarks(file))
    checkButton.place(x=20,y=123)
```

## Testing

I then make a simple test that should output 4 marks; however, it outputs a much larger number.





I fix the error by creating a function similar to my openLogin function, which reads the login.csv file row by row.

```
def checkMarks(file):
    marks=-1

    contents=open(file)
    fileContents=(contents)
    next(contents)

    for row in fileContents:
        marks=marks+1

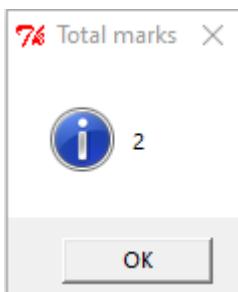
    messagebox.showinfo("Total marks",marks)
```

## Testing

I create another simple test.



However, I get the wrong number



I fix this by changing the starting value of marks as it does not read the first line, which is the question and answer headings.

```
def checkMarks(file):
    marks=0

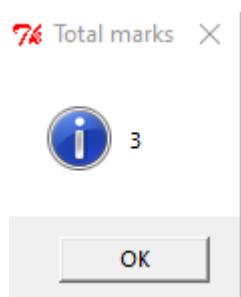
    contents=open(file)
    fileContents=(contents)
    next(contents)

    for row in fileContents:
        marks=marks+1

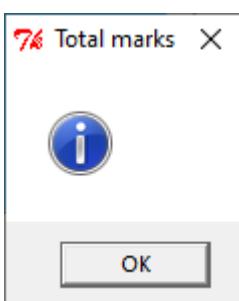
    messagebox.showinfo("Total marks",marks)
```

## Testing

Now the function outputs the correct number of marks



However, if no questions or answers are inputted. The message box does not display any message. To fix this I will make an if statement to output a show info message box if the marks is larger than 0 and output a show error message box if the marks is equal to 0, along with an appropriate message.



I make the changes.

```
def checkMarks(file):
    marks=0

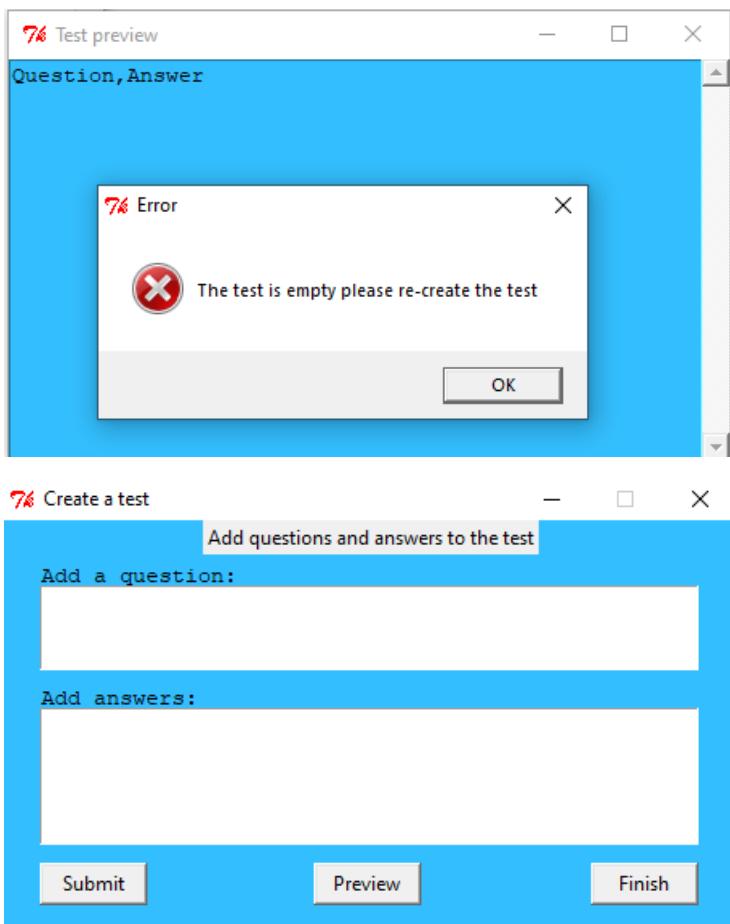
    contents=open(file)
    fileContents=(contents)
    next(contents)

    for row in fileContents:
        marks=marks+1

    if marks!=0:
        messagebox.showinfo("Total marks",marks)
    else:
        messagebox.showerror("Error","The test is empty please re-create the test")
        openCreatedTest(file)
```

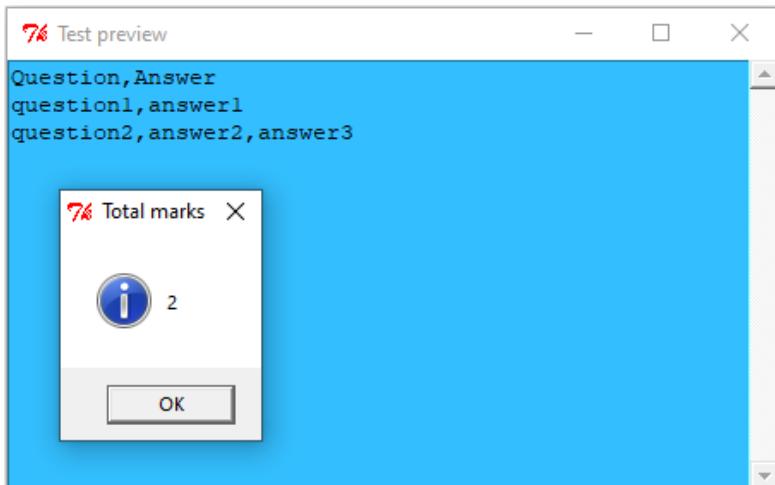
## Testing

When an empty test is created the message appears and the create a test window opens again. This also keeps the created test, so the test name does not have to be re-entered. **Otherwise, the user will need to re-enter the name of the test.**



## Testing

When a test is created with questions and answers, the marks are still shown correctly.



## Submit button / Retrieve grade boundaries and save them

Next, I will create the function to input the grades.

For the first function, which should create a new .csv file to hold the grades. This will be based off the testFileCreator function.

```
submitButton=Button(gradeWindow,text="Submit",command=lambda: gradeFileCreator(file))
submitButton.place(x=230,y=123)

def gradeFileCreator(testName):
    rootFile=os.path.abspath(os.curdir)
    gradeFileName=testName.strip(".csv")
    dstFile=gradeFileName+" - grades"+".csv"
    rootFile=rootFile+"\empty.csv"

    shutil.copy2(rootFile,dstFile)
```

 test	25/01/2023 12:05	Microsoft Excel C...	1 KB
 test - grades	06/12/2022 14:57	Microsoft Excel C...	1 KB

This function runs with no errors

## Grade file creator function

The next function will open the new test and add the correct headers for the grades. This will be based off the openCreatedTest function.

### Error

I make the function but receive an error.

```
def openGradeFile(file):
    csvFile=open(file,'r+')
    csvFile.write('A,B,C,D,E,F\n')
    file=csvFile.read()

    for item in file:
        csvFile.write('{A},{B},{C},{D},{E},{F}'.format(**item))
        csvFile.write('\n')
    csvFile.close()

def gradeFileCreator(testName):
    rootFile=os.path.abspath(os.curdir)
    gradeFileName=testName.strip(".csv")
    dstFile=gradeFileName+" - grades"+".csv"
    rootFile=rootFile+"\empty.csv"

    shutil.copy2(rootFile,dstFile)

    openGradeFile(testName)
```

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 122, in <lambda>
    submitButton=Button(gradeWindow,text="Submit",command=lambda: gradeFileCreate
or(file))
  File "N:\GUI.py", line 46, in gradeFileCreator
    openGradeFile(testName)
  File "N:\GUI.py", line 34, in openGradeFile
    csvFile.write('{A},{B},{C},{D},{E},{F}'.format(**item))
TypeError: format() argument after ** must be a mapping, not str

```

## Testing

I realise I that openGradeFile has the wrong argument, so I correct it. The program now correctly makes the file

```

def gradeFileCreator(testName):
    rootFile=os.path.abspath(os.curdir)
    gradeFileName=testName.strip(".csv")
    dstFile=gradeFileName+" - grades"+".csv"
    rootFile=rootFile+"\empty.csv"

    shutil.copy2(rootFile,dstFile)

    openGradeFile(dstFile)

```

	A	B	C	D	E	F	G
1	A	B	C	D	E	F	
2							

## Get grades function

Next, I will make a function to get the grades from the input boxes. This will be based off my appendDetails function

```

def getGrades(A,B,C,D,E,F,file):
    grades=[]
    a=A.get()
    b=B.get()
    c=C.get()
    d=D.get()
    e=E.get()
    f=F.get()
    gradesData={'A':a,'B':b,'C':c,'D':d,'E':e,'F':f}
    grades.append(gradesData)

    if len(a)!=0 and len(b)!=0 and len(c)!=0 and len(d)!=0 and len(e)!=0 and len(f)!=0:
        print("test")
    else:
        messagebox.showerror("Grades not added","Please fill all input boxes and try again")

```

The program runs with no errors, but the if statement does not produce an output. To try to fix this I modify the .get() parameters. However, no output is given still.

```

def getGrades(A,B,C,D,E,F,file):
    grades=[]
    a=A.get(1.0,'end-1c')
    b=B.get(1.0,'end-1c')
    c=C.get(1.0,'end-1c')
    d=D.get(1.0,'end-1c')
    e=E.get(1.0,'end-1c')
    f=F.get(1.0,'end-1c')
    gradesData={'A':a,'B':b,'C':c,'D':d,'E':e,'F':f}
    grades.append(gradesData)

    if len(a)!=0 and len(b)!=0 and len(c)!=0 and len(d)!=0 and len(e)!=0 and len(f)!=0:
        print("test")
    else:
        messagebox.showerror("Grades not added","Please fill all input boxes and try again")

```

To fix the problem, I finish the call function to call getGrades. I also add new parameters and arguments to ensure that when it is called there is no errors.

```

def openGradeFile(file,A,B,C,D,E,F):
    csvFile=open(file,'r+')
    csvFile.write('A,B,C,D,E,F\n')
    file=csvFile.read()

    for item in file:
        csvFile.write('{A},{B},{C},{D},{E},{F}'.format(**item))
        csvFile.write('\n')
    csvFile.close()

    getGrades(A,B,C,D,E,F,file)

```

```

submitButton=Button(gradeWindow,text="Submit",command=lambda: gradeFileCreator(file,entryAlabel,entryBlabel,entryClabel,entryDlabel,entryElabel,entryFlabel))
submitButton.place(x=230,y=123)

```

```

def gradeFileCreator(testName,A,B,C,D,E,F):
    rootFile=os.path.abspath(os.curdir)
    gradeFileName=testName.strip(".csv")
    dstFile=gradeFileName+" - grades"+".csv"
    rootFile=rootFile+"\empty.csv"

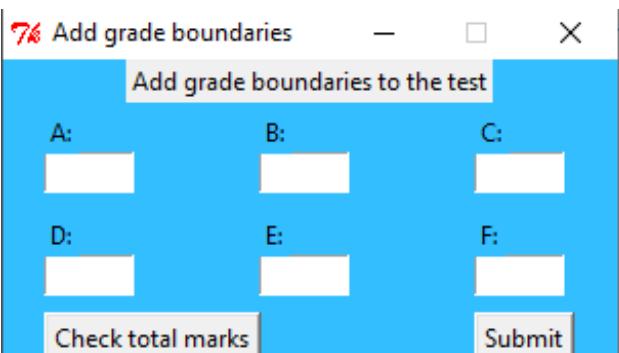
    shutil.copy2(rootFile,dstFile)

    openGradeFile(dstFile,A,B,C,D,E,F)

```

## Error

However, when the final submit button is pressed an error is given.



```

    se()" for more information.
RESTART =====
RESTART =====
RESTART =====
RESTART =====
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 133, in <lambda>
    submitButton=Button(gradeWindow,text="Submit",command=lambda: gradeFileCreate
or(file,entryAlabel,entryBlabel,entryClabel,entryDlabel,entryElabel,entryFlabel)
)
NameError: global name 'entryAlabel' is not defined

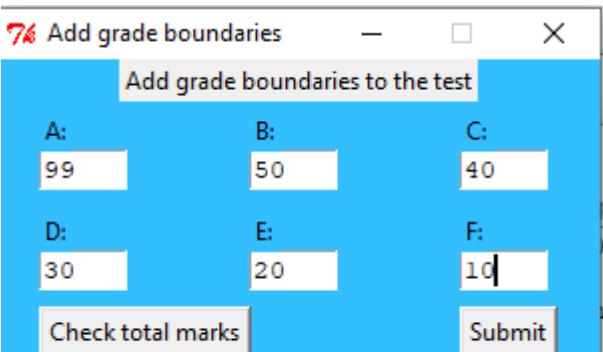
```

I realise I mixed up the variable names for the label widget and text widget and correct this

```
gradeFileCreator(file, entryAgrade, entryBgrade, entryCgrade, entryDgrade, entryEgrade, entryFgrade)
```

## Testing

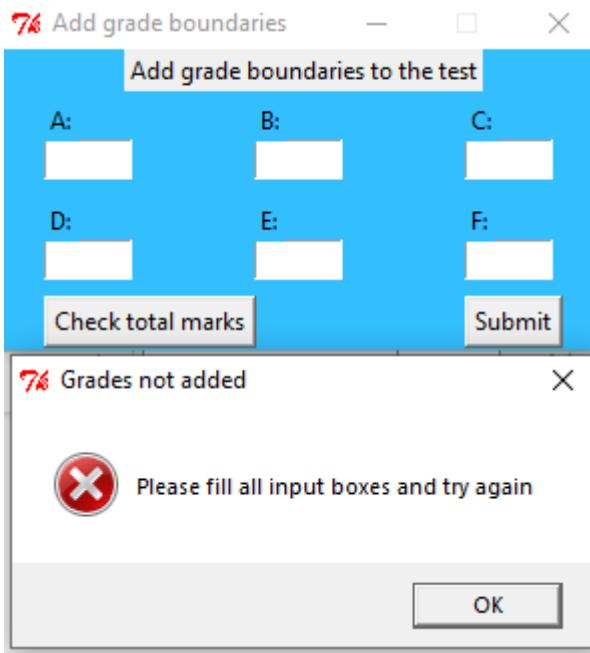
The program runs with no errors and the if statement also works



```

    >>>
test

```



## Save grades function

Next, I will make a function to save these grades to the grades csv file. This will be based off my saveTestData function.

I make the function and link it to execute if all input boxes are not empty.

```
def saveGrades(grades, file, gradesData):
    try:
        fileHandle=open(file,'r+')
        fileContent=fileHandle.read()
        if fileContent.strip()=='':
            fileHandle.write('A,B,C,D,E,F\n')
        for item in grades:
            fileHandle.write('{A},{B},{C},{D},{E},{F}'.format(**gradesData))
            fileHandle.write('\n')
        messagebox.showinfo("Success","Grades saved successfully")

        fileHandle.close()
    except OSError:
        messagebox.showerror("Not saved","Grades not saved successfully, please try again")

if len(a)!=0 and len(b)!=0 and len(c)!=0 and len(d)!=0 and len(e)!=0 and len(f)!=0:
    saveGrades(grades,file,gradesData)
```

## Error

However, when the program is run an error is received.

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 148, in <lambda>
    submitButton=Button(gradeWindow,text="Submit",command=lambda: gradeFileCreator(file,entryAgrade,entryBgrade,entryCgrade,entryDgrade,entryEgrade,entryFgrade))
)
  File "N:\GUI.py", line 79, in gradeFileCreator
    openGradeFile(dstFile,A,B,C,D,E,F)
  File "N:\GUI.py", line 69, in openGradeFile
    getGrades(A,B,C,D,E,F,file)
  File "N:\GUI.py", line 55, in getGrades
    saveGrades(grades,file,gradesData)
  File "N:\GUI.py", line 30, in saveGrades
    fileHandle=open(file,'r+')
IOError: [Errno 2] No such file or directory: ''
```

I realise that the file name is overwritten in this read statement, so I change variable name. **Otherwise, the file cannot be accessed.**

```
def openGradeFile(file,A,B,C,D,E,F):
    csvFile=open(file,'r+')
    csvFile.write('A,B,C,D,E,F\n')
    file=csvFile.read()

    for item in file:
        csvFile.write('{A},{B},{C},{D},{E},{F}'.format(**item))
        csvFile.write('\n')
    csvFile.close()

    getGrades(A,B,C,D,E,F,file)

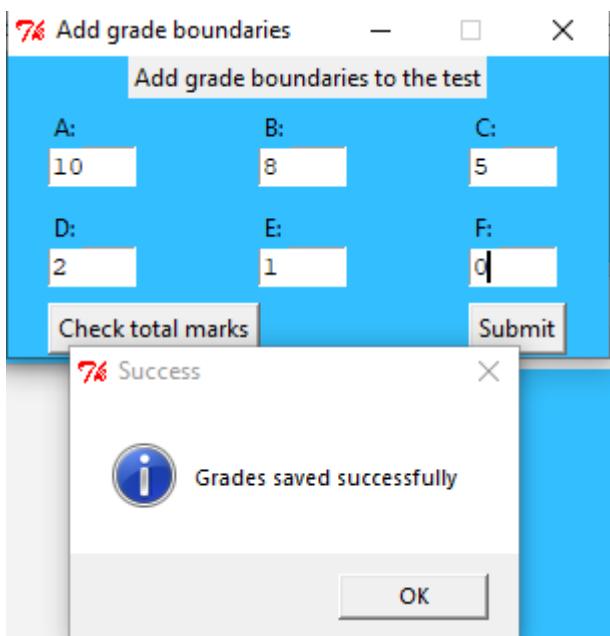
def openGradeFile(file,A,B,C,D,E,F):
    csvFile=open(file,'r+')
    csvFile.write('A,B,C,D,E,F\n')
    fileRead=csvFile.read()

    for item in fileRead:
        csvFile.write('{A},{B},{C},{D},{E},{F}'.format(**item))
        csvFile.write('\n')
    csvFile.close()

    getGrades(A,B,C,D,E,F,file)
```

## Testing

When I run the program, no errors are received, and the grades are added to the grades file successfully.



	A	B	C	D	E	F
1	A	B	C	D	E	F
2	10	8	5	2	1	0
3						
4						

I also add gradeWindow to every parameter and argument in all functions up to saveGrades(), then I use destroy to close the window preventing the user from adding more inputs to the file.

```
def saveGrades(grades, file, gradesData, gradeWindow):
    try:
        fileHandle=open(file,'r+')
        fileContent=fileHandle.read()
        if fileContent.strip()=='':
            fileHandle.write('A,B,C,D,E,F\n')
        for item in grades:
            fileHandle.write('{A},{B},{C},{D},{E},{F}'.format(**gradesData))
            fileHandle.write('\n')
        messagebox.showinfo("Success","Grades saved successfully")
        gradeWindow.destroy()
        fileHandle.close()

    except OSError:
        messagebox.showerror("Not saved","Grades not saved successfully, please try again")
```

## Delete last button

The final function I need to add is a delete button that will delete the last line of a test when adding details to the test.

I first create a new button and rearrange other buttons, so they all fit on the GUI. **Otherwise, not all buttons are visible.**

```

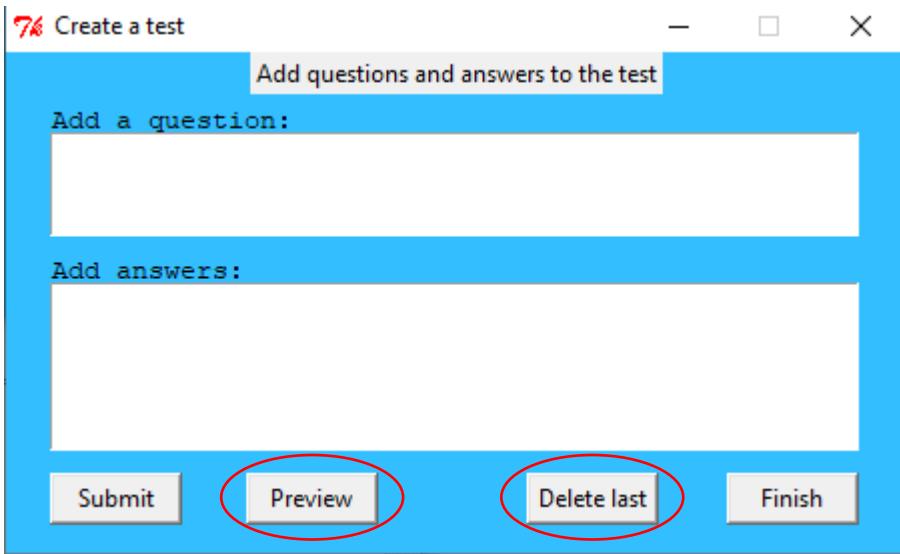
submitButton=Button	appendWindow, text="Submit", width=8, command=lambda: appendDetails(entryAddQuestion, entryAddAnswer, testFile))
submitButton.place(x=22, y=210)

previewButton=Button	appendWindow, text="Preview", width=8, command=lambda: testPreview(testFile))
previewButton.place(x=120, y=210)

deleteButton=Button	appendWindow, text="Delete last", width=8)
deleteButton.place(x=260, y=210)

finishButton=Button	appendWindow, text="Finish", width=8, command=lambda: endTest.appendWindow, testFile))
finishButton.place(x=360, y=210)

```



Next, I make a function to delete the last line of the csv file, this function will specifically read the csv file into a format that can be used by my saveTestData() function, which will write the new details by overwriting the previous details (by opening the csv file in the write mode as oppose to the read/append mode) compared to the current purpose of appending details.

First, I make a function to format the data appropriately

```

def deleteLast(testFile):
    csvFile=open(testFile, 'r+')
    csvFileContent=(csvFile)
    next(csvFile)

    csvList=[]

    for row in csvFileContent:
        csv=row.strip().split(",")
        heading=['Question', 'Answer']
        data=zip(heading,csv)
        DataDict=dict(data)
        csvList.append(DataDict)

    csvList.pop()
    saveTestData(csvList,testFile,DataDict,'w+')

```

Next, I change the parameters and arguments of both the saveTestData() function and every instance that it is called so that my fourth parameter is suitable when calling the function. I also change the message box depending on if questions and answers are deleted or added

```

def saveTestData(csv,file,csvData,mode):
    try:
        fileHandle=open(file,mode)
        fileContent=fileHandle.read()
        if fileContent.strip()=='':
            fileHandle.write('Question,Answer\n')
        for item in csv:
            fileHandle.write('{Question},{Answer}'.format(**csvData))
            fileHandle.write('\n')

    if mode=='r+':
        messagebox.showinfo("Success","Question and answers added successfully")
    elif mode=='w+':
        messagebox.showinfo("Success","The previous entries have been removed successfully")

    fileHandle.close()
    except OSError:
        print('Can\'t write to file')

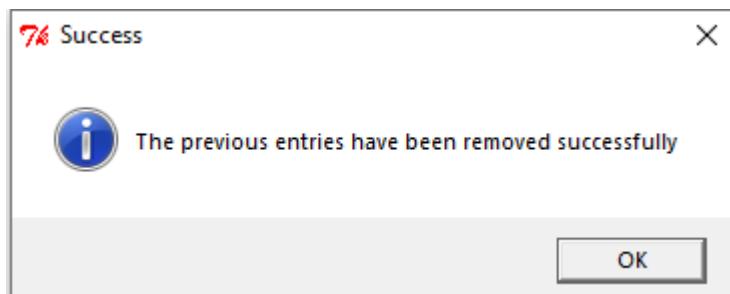
def appendDetails(inputtedQuestion,inputtedAnswers,testFile):
    csv=[]
    question=inputtedQuestion.get(1.0,'end-1c')
    answers=inputtedAnswers.get(1.0,'end-1c')
    csvData={'Question':question,'Answer':answers}
    csv.append(csvData)

    if len(question)!=0 and len(answers)!=0:
        saveTestData(csv,testFile,csvData,'r+')
    else:
        messagebox.showerror("Questions and answers not added","Both text boxes must be filled, please try again")

```

## Testing

The program runs with no errors when using both buttons.



## Error

However, an error message appears when the delete last button is pressed, when the csv file has no more questions and answers to delete.

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 268, in <lambda>
    deleteButton=Button	appendWindow,text="Delete last",width=8,command=lambda:
deleteLast(testFile)
  File "N:\GUI.py", line 224, in deleteLast
    next(csvFile)
StopIteration

```

Although, the error message has no effect on the program, I use an if statement to ensure that the list is only popped if it is not empty, and I add an appropriate error message if the list is empty.

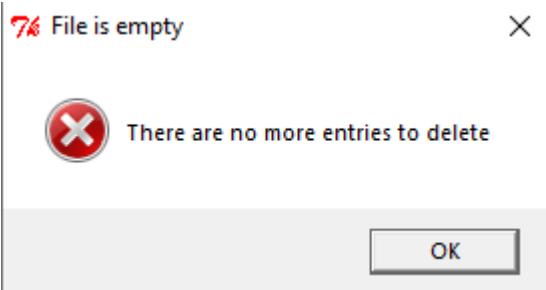
```

def deleteLast(testFile):
    csvFile=open(testFile,'r+')
    csvFileContent=(csvFile)
    next(csvFile)

    csvList=[]

    for row in csvFileContent:
        csv=row.strip().split(",")
        heading=['Question','Answer']
        data=zip(heading,csv)
        DataDict=dict(data)
        csvList.append(DataDict)

    if len(csvList)!=0:
        csvList.pop()
        saveTestData(csvList,testFile,DataDict,'w+')
    else:
        messagebox.showerror("File is empty","There are no more entries to delete")
    
```



This is the end of my first prototype for my create a test process. Next, I will work on the functionality for my complete a test button.

## Complete a test

### Name the test GUI

First, the user needs to enter the name of a test. The user will do this using a GUI, due to the similarity between this and my nameTestFile function. I will modify this current function into a function that can switch between entering the name of a test file to complete and entering the name of a test file to create.

```

def nameTestFile(title,function):
    fileCreator=Tk()
    fileCreator.geometry("450x42")
    fileCreator.title(title)
    fileCreator.configure(bg="#33BFFF")

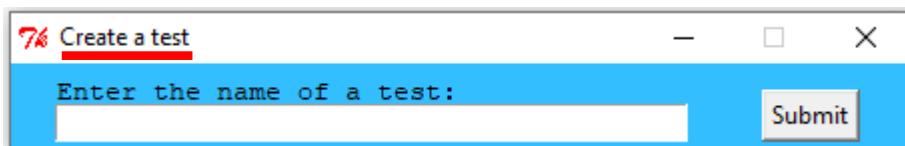
    fileCreator.resizable(False,False)

    labelCreator=Label(fileCreator,text="Enter the name of a test:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
    labelCreator.place(x=22,y=5)

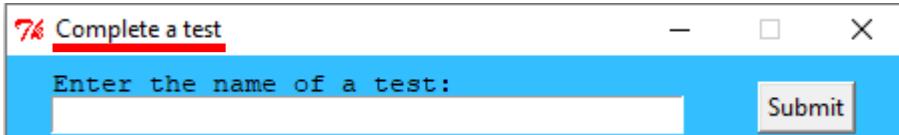
    nameOfTestBox=Entry(fileCreator,width=52)
    nameOfTestBox.place(x=22,y=20)

    if function=="Create":
        submitButton=Button(fileCreator,text="Submit",command=lambda:testFileCreator(nameOfTestBox,fileCreator))
        submitButton.place(x=375,y=13)
    elif function=="Complete":
        submitButton=Button(fileCreator,text="Submit")
        submitButton.place(x=375,y=13)
    
```

```
buttonCreate=Button(sec,text="Create a test",command=lambda: nameTestFile("Create a test","Create"))
buttonCreate.place(x=10,y=150)
```



```
buttonComplete=Button(sec,text="Complete a test",command=lambda: nameTestFile("Complete a test","Complete"))
buttonComplete.place(x=10, y=20)
```



## Testing

The program still runs with no errors and can work for both the create a test button and the complete a test button.

### Check file exists function

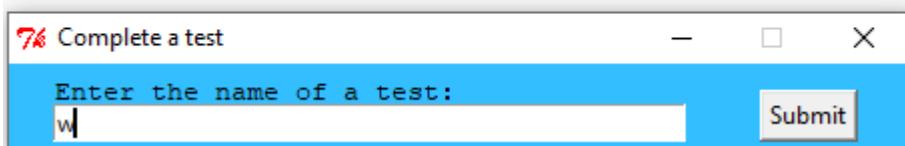
Next, I make a function to check if the name of the file enters exists.

```
def checkFileName(testName):
    testName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    file=rootFile+"\\"+testName+".csv"

    print(file)

    print(os.path.isfile(file))
```

```
N:\\a.csv
True
N:\\w.csv
False
```



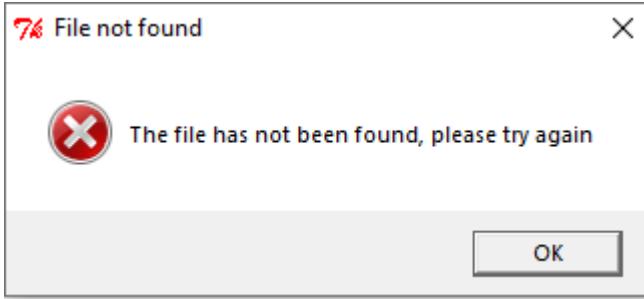
I add an if statement, which I will use to proceed to the next function if the file exists, otherwise an error message will appear. Without this the user would not be able to know if their input has not been accepted.

```

def checkFileName(testName):
    testName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    file=rootFile+"\\"+testName+".csv"

    if os.path.isfile(file)==True:
        #next function
        return(file)
    elif os.path.isfile(file)==False:
        messagebox.showerror("File not found","The file has not been found, please try again")

```



## Read test file function

Next, I will make a function to read the csv file.

I realise reading the test file in the necessary format is already a function in deleteLast(), therefore I will turn this transform this function to fulfil both purposes instead of using two functions. This will save space and memory and will simplify the code.

I add the necessary if statement and change the parameters and arguments

```

def deleteLast(testFile):
    csvFile=open(testFile,'r+')
    csvFileContent=(csvFile)
    next(csvFile)

    csvList=[]

    for row in csvFileContent:
        csv=row.strip().split(",")
        heading=['Question','Answer']
        data=zip(heading,csv)
        DataDict=dict(data)
        csvList.append(DataDict)

    csvList.pop()
    saveTestData(csvList,testFile,DataDict,'w+')

```

```

def readTestFile(testFile, mode):
    csvFile=open(testFile, 'r+')
    csvFileContent=(csvFile)
    next(csvFile)

    csvList=[]

    for row in csvFileContent:
        csv=row.strip().split(",")
        heading=['Question','Answer']
        data=zip(heading,csv)
        DataDict=dict(data)
        csvList.append(DataDict)

    if mode=="Delete":
        if len(csvList)!=0:
            csvList.pop()
            saveTestData(csvList,testFile,DataDict, 'w+')
        else:
            messagebox.showerror("File is empty","There are no more entries to delete")
    elif mode=="Read":
        #next function
        return(mode)

```

```

deleteButton=Button	appendWindow, text="Delete last", width=8, command=lambda: readTestFile(testFile, "Delete"))
deleteButton.place(x=260, y=210)

```

## Error

I receive an error when I enter the name of a test file to complete

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 378, in <lambda>
    submitButton=Button(fileCreator, text="Submit", command=lambda:checkFileName(n
ameOfFileBox))
  File "N:\GUI.py", line 60, in checkFileName
    readTestFile(testFile)
TypeError: readTestFile() takes exactly 2 arguments (1 given)

```

I fix this by adding the second argument to the checkFileName() function.

```

def checkFileName(testName):
    testName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    testFile=testName+".csv"
    file=rootFile+"\\"+testName+".csv"

    if os.path.isfile(file)==True:
        readTestFile(testFile, "Read")
    elif os.path.isfile(file)==False:
        messagebox.showerror("File not found","The file has not been found, please try again")

```

## Test completion GUI

Next, I will create the GUI. I also modify some previous functions' parameters and arguments in order to pass the `testName` variable to the GUI to be used as the window's title.

```
def checkFileName(testName):
    testName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    file=rootFile+"\\"+testName+".csv"

    if os.path.isfile(file)==True:
        readTestFile(testName, "Read")
    elif os.path.isfile(file)==False:
        messagebox.showerror("File not found",

def readTestFile(testName,mode):
    testFile=testName+".csv"
    csvFile=open(testFile,'r+')
    csvFileContent=(csvFile)
    next(csvFile)

    csvList=[]

def testCompletion(testName):
    completeWindow=Tk()
    completeWindow.geometry("450x250")
    completeWindow.title(testName)
    completeWindow.configure(bg="#33BFFF")

    completeWindow.resizable(False,False)

    scrollbar=Scrollbar(completeWindow)
    scrollbar.pack(side=RIGHT,fill=Y)

    questionLabel=Label(completeWindow,text="Question:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
    questionLabel.place(x=20,y=0)

    questionBox=Text(completeWindow,width=50,height=4,wrap=WORD)
    questionBox.place(x=20,y=20)

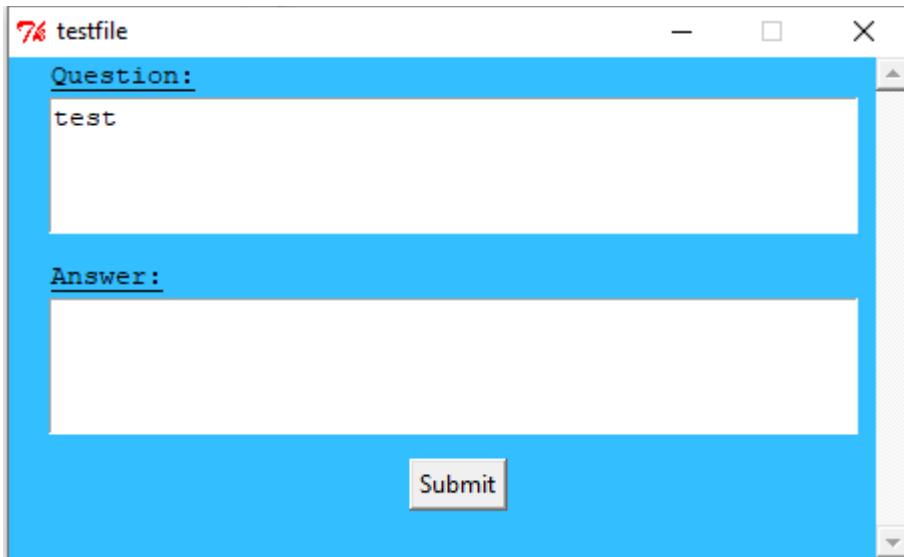
    question="test"
    questionBox.insert(END,question)

    questionBox.config(yscrollcommand=scrollbar.set,state=DISABLED)
    scrollbar.config(command=questionBox.yview)

    answerLabel=Label(completeWindow,text="Answer:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
    answerLabel.place(x=20,y=100)

    answerBox=Text(completeWindow,width=50,height=4,wrap=WORD)
    answerBox.place(x=20,y=120)

    submitButton=Button(completeWindow,text="Submit")
    submitButton.place(x=200,y=200)
```



## Update question function

### Error

Next, I will work on a system to update the question and check the answer whenever the 'Submit' button is pressed. **Otherwise, the next question will not appear, and the answer cannot be checked.**

This function should add the questions to the question box using a loop. If this works, then I will then modify this function to only add questions when the submit button is pressed. However, I receive an error message.

```
def updateQuestion(testList,questionBox):
    questionBox.config(state=NORMAL)
    for i in testList():
        question=i['Question']
        print(question)
        questionBox.insert(END,question)
    questionBox.config(state=DISABLED)

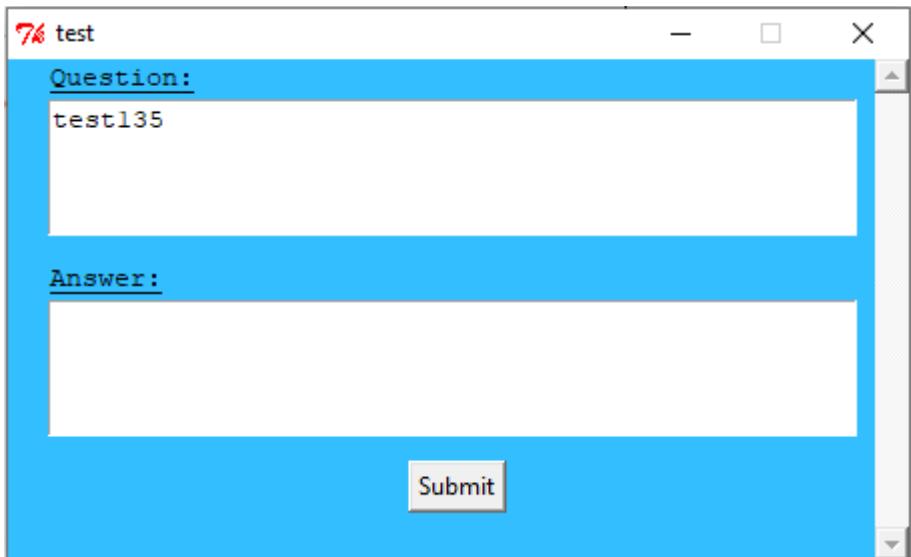
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 102, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuest
ion(testList,questionBox))
  File "N:\GUI.py", line 67, in updateQuestion
    for i in testList():
TypeError: 'list' object is not callable
```

To fix this error I correct the for loop from `testList()` to `testList`

```
def updateQuestion(testList,questionBox):
    questionBox.config(state=NORMAL)
    for i in testList:
        question=i['Question']
        print(question)
        questionBox.insert(END,question)
    questionBox.config(state=DISABLED)
```

## Testing

The program now correctly adds the questions from the test file.



	A	B	C
1	Question	Answer	
2	1	2	
3	3	4	
4	5	6	
5			
6			
7			

Now I need to both clear the question box each time a question is to be added, and only add a question each time the submit button is pressed.

To do this, I modify the updateQuestion function turning it into the new readAllQuestions function. This will allow all questions to be read into an array, which will then be passed into the test completion GUI, where it will be read into the 'Submit' button, which will execute another function which will update the question. This allows a new question to only appear once the submit button is pressed. **Otherwise, the user cannot submit their answer and view the next question when they are ready.**

```

def readTestFile(testName, mode):
    testFile=testName+".csv"
    csvFile=open(testFile, 'r+')
    csvFileContent=(csvFile)
    next(csvFile)

    csvList=[]

    for row in csvFileContent:
        csv=row.strip().split(",")
        heading=['Question', 'Answer']
        data=zip(heading,csv)
        DataDict=dict(data)
        csvList.append(DataDict)

    if mode=="Delete":
        if len(csvList)!=0:
            csvList.pop()
            saveTestData(csvList,testFile,DataDict,'w+')
        else:
            messagebox.showerror("File is empty","There are no more entries")
    elif mode=="Read":
        readAllQuestions(csvList,testName)

def readAllQuestions(testList,testName):
    questions={}
    for i in testList:
        question=i['Question']
        questions[i]=question

    testCompletion(testName,testList)

```

## Error

However, I receive an error.

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 401, in <lambda>
    submitButton=Button(fileCreator,text="Submit",command=lambda:checkFileName
ameOfFileTextBox))
  File "N:\GUI.py", line 59, in checkFileName
    readTestFile(testName,"Read")
  File "N:\GUI.py", line 46, in readTestFile
    readAllQuestions(csvList,testName)
  File "N:\GUI.py", line 68, in readAllQuestions
    questions[i]=question
TypeError: list indices must be integers. not dict

```

I fix this by using the correct methods for adding to an array

```

def readAllQuestions(testList,testName):
    questions=[]
    for i in testList:
        question=i['Question']
        questions.append(question)

    print(questions)
    testCompletion(testName,testList)

```

## Testing

The program now runs with no errors and the array is printed correctly to show that the array has been added to correctly.

```
['1', '3', '5']
```

Next, I need a function to clear the question box and add the new question from the array.

```

count=-1
question="test"
questionBox.insert(END,question)

questionBox.config(yscrollcommand=scrollbar.set,state=DISABLED)
scrollbar.config(command=questionBox.yview)

answerLabel=Label(completeWindow,text="Answer:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
answerLabel.place(x=20,y=100)

answerBox=Text(completeWindow,width=50,height=4,wrap=WORD)
answerBox.place(x=20,y=120)

submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,count+1))
submitButton.place(x=200,y=200)

def updateQuestionBox(questionBox,questions,count):
    questionBox.config(state=NORMAL)
    questionBox.delete(0,END)

    question=questions[count]

    questionBox.insert(END,question)

    questionBox.config(state=DISABLED)

```

## Error

However, I receive an error.

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 114, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuest
ionBox(questionBox,questions,count+1))
  File "N:\GUI.py", line 75, in updateQuestionBox
    questionBox.delete(0,END)
  File "C:\Python32\lib\tkinter\__init__.py", line 2837, in delete
    self.tk.call(self._w, 'delete', index1, index2)
_tkinter.TclError: bad text index "0"

```

To fix this I modify the start index of the delete() method to the correct format

```
def updateQuestionBox(questionBox, questions, count):
    questionBox.config(state=NORMAL)
    questionBox.delete(1.0, END)

    question=questions[count]

    questionBox.insert(END, question)

    questionBox.config(state=DISABLED)
```

## Testing

Now the program correctly run, however the first time the submit button is pressed it displays the first question, as it should, but pressing the submit button after this does not display any more questions. To fix this I will need to modify the use of iteration for the array

```
def testCompletion(testName,testList,questions):
    completeWindow=Tk()
    completeWindow.geometry("450x250")
    completeWindow.title(testName)
    completeWindow.configure(bg="#33BFFF")

    completeWindow.resizable(False,False)

    scrollbar=Scrollbar(completeWindow)
    scrollbar.pack(side=RIGHT,fill=Y)

    questionLabel=Label(completeWindow,text="Question:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
    questionLabel.place(x=20,y=0)

    questionBox=Text(completeWindow,width=50,height=4,wrap=WORD)
    questionBox.place(x=20,y=20)

    questionBox.insert(END,questions[0])

    questionBox.config(yscrollcommand=scrollbar.set,state=DISABLED)
    scrollbar.config(command=questionBox.yview)

    answerLabel=Label(completeWindow,text="Answer:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
    answerLabel.place(x=20,y=100)

    answerBox=Text(completeWindow,width=50,height=4,wrap=WORD)
    answerBox.place(x=20,y=120)

    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions))
    submitButton.place(x=200,y=200)
```

```
def updateQuestionBox(questionBox, questions):
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0, 'end-1c')

    questionBox.delete(1.0, END)

    i=0
    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            question=questions[i+1]
            break

    questionBox.insert(END, question)

    questionBox.config(state=DISABLED)
```

This function works by inserting the first question when defining the GUI in testCompletion(), then the updateQuestionBox() function reads the current question and finds the index in the array, then it increments it by 1 to get the next question and adds it to the question box.

## Error

The program runs correctly until you press the submit button when there are no more questions in the array (the test has been completed), this will be when the test ends and a summary screen appears.

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 121, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions))
  File "N:\GUI.py", line 85, in updateQuestionBox
    question=questions[i+1]
IndexError: list index out of range
```

To fix this I modify the range of the while loop and move the insert() method to prevent a future error. Otherwise, an index error would appear if a question that hasn't been defined is inserted

```
def updateQuestionBox(questionBox,questions):
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0,'end-1c')

    questionBox.delete(1.0,END)

    i=0
    while i!=len(questions)-1:
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            question=questions[i+1]
            questionBox.insert(END,question)
            break

    questionBox.config(state=DISABLED)
```

## Check answer function

Next, I need to make a function to receive the answer and check if it is correct. **Otherwise, the user would not know if their answer were correct**

```
def checkAnswer(answerBox,answers,i):
    answer=answerBox.get(1.0,END)

    print(answers[i])
    print(answer)

    if answer==answers[i]:
        print("correct")
    elif answer!=answers[i]:
        print("wrong")
```

## Testing

However, the program never outputs correct if the inputted answer is correct

```
>>>
answer1
answer2
answer3
['question1', 'question2', 'question3']
['answer1', 'answer2', 'answer3']
answer1
answer1

wrong
answer2
answer2

wrong
```

To fix this I change the index of the .get() method. By reducing the end index from end to end-1c, I avoid the trailing new line that tkinter adds to the end of text widgets automatically

```
"""
def checkAnswer(answerBox, answers, i):
    answer=answerBox.get(1.0, 'end-1c')

    print(answers[i])
    print(answer)

    if answer==answers[i]:
        print("correct")
    elif answer!=answers[i]:
        print("wrong")
```

Now, the answers are corrected properly. However, the last answer is not received by the program. **Without this, the test result cannot be properly evaluated.**

```
answer1
answer2
answer3
['question1', 'question2', 'question3']
['answer1', 'answer2', 'answer3']
answer1
answer1
correct
answer2
answer2
correct
```

## Error

To fix this I change the length of the loop again, but I receive the index error as stated earlier

```
def updateQuestionBox(questionBox, questions, answerBox, answers):
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0, 'end-1c')

    questionBox.delete(1.0, END)

    i=0
    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            checkAnswer(answerBox, answers, i)
            question=questions[i+1]
            questionBox.insert(END, question)
            break

    questionBox.config(state=DISABLED)|
```

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 139, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers))
  File "N:\GUI.py", line 104, in updateQuestionBox
    question=questions[i+1]
IndexError: list index out of range

```

To fix this I add another if statement, preventing the use of an index out of range

```

def updateQuestionBox(questionBox,questions,answerBox,answers):
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0,'end-1c')

    questionBox.delete(1.0,END)

    i=0
    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            checkAnswer(answerBox,answers,i)
            if i+1<len(questions):
                question=questions[i+1]
                questionBox.insert(END,question)
                break
            break

    questionBox.config(state=DISABLED)

```

Now, all answers are read

(Program output)

```

answer1
answer2
answer3
['question1', 'question2', 'question3']
['answer1', 'answer2', 'answer3']
answer1
answer1
correct
answer2
answer2
correct
answer3
answer3
correct

```

I move the code that gets the answer from the answer box, from the checkAnswer command to the updateQuestionBox command as it makes more sense. I also modify the parameters and arguments appropriately.

I also add an error message that appears when the answer box is empty and also add an extra if statement that stops outputting questions once the test is complete. **Otherwise, the program would not know when the test has ended and it will not know if an answer is valid.**

```
def checkAnswer(answer,answers,i):
    print(answers[i])
    print(answer)

    if answer==answers[i]:
        print("correct")
    elif answer!=answers[i]:
        print("wrong")

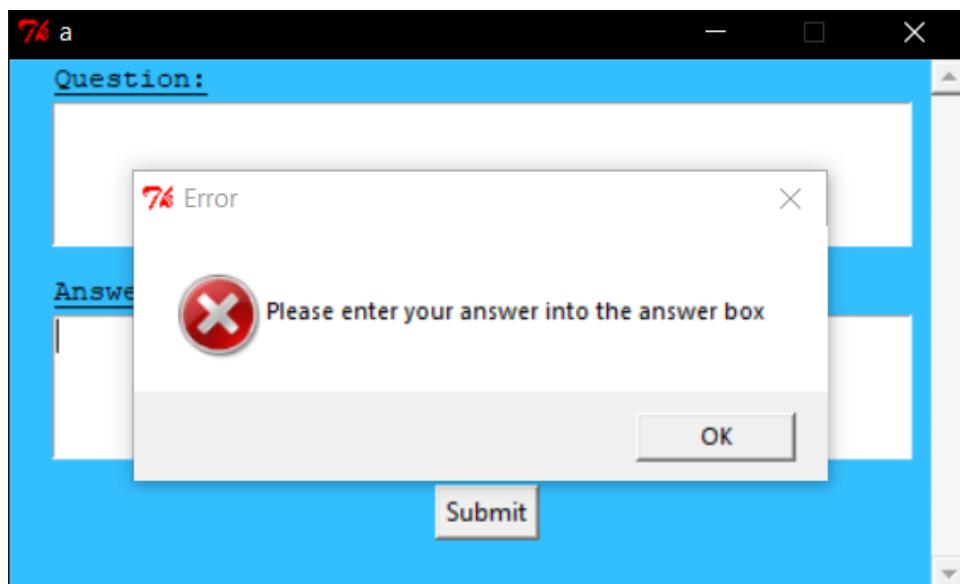
def updateQuestionBox(questionBox,questions,answerBox,answers):
    answer=answerBox.get(1.0,'end-1c')
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0,'end-1c')

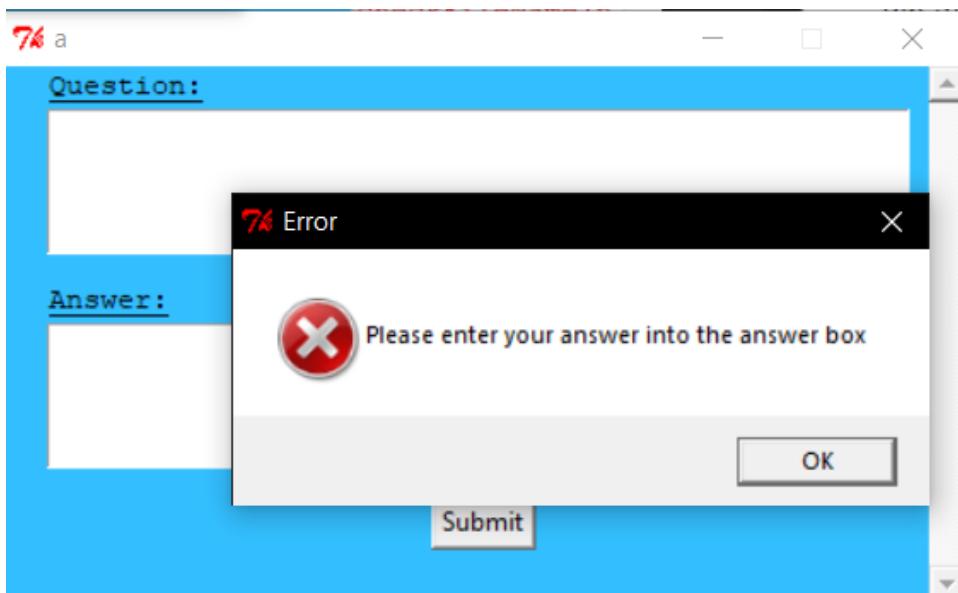
    questionBox.delete(1.0,END)

    i=0
    while i!=len(questions) and answer!="":
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            checkAnswer(answer,answers,i)
            if i+1<len(questions):
                question=questions[i+1]
                questionBox.insert(END,question)
                break
            break
    if answer=="":
        messagebox.showerror("Error","Please enter your answer into the answer box")

    questionBox.config(state=DISABLED)
```



I find out that attempting to enter a blank answer into the answer box removes the current question. **Without this, the user cannot re-attempt to answer the question.**



I try to fix this by inserting the question into the question box after the error is gone, however this leads to only the first question being displayed as *i* is set to 0, every time the while loop ends.

```
i=0
while i!=len(questions) and answer!="":
    if currentQuestion!=questions[i]:
        i=i+1
    elif currentQuestion==questions[i]:
        checkAnswer(answer,answers,i)
        if i+1<len(questions):
            #change question variable name
            question=questions[i+1]
            questionBox.insert(END,question)
            break
    #test|
    elif i==len(questions):
        print("end of test")
        break

if answer=="":
    messagebox.showerror("Error","Please enter your answer into the answer box")
    questionBox.insert(END,questions[i])
    questionBox.config(state=DISABLED)
```

I fix this by removing the condition in the while loop that the answer is not blank. This prevents the loop from ending and therefore the current value of *i* is preserved throughout the whole function and not just within the loop.

```
def updateQuestionBox(questionBox,questions,answerBox,answers):
    answer=answerBox.get(1.0,'end-1c')
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0,'end-1c')

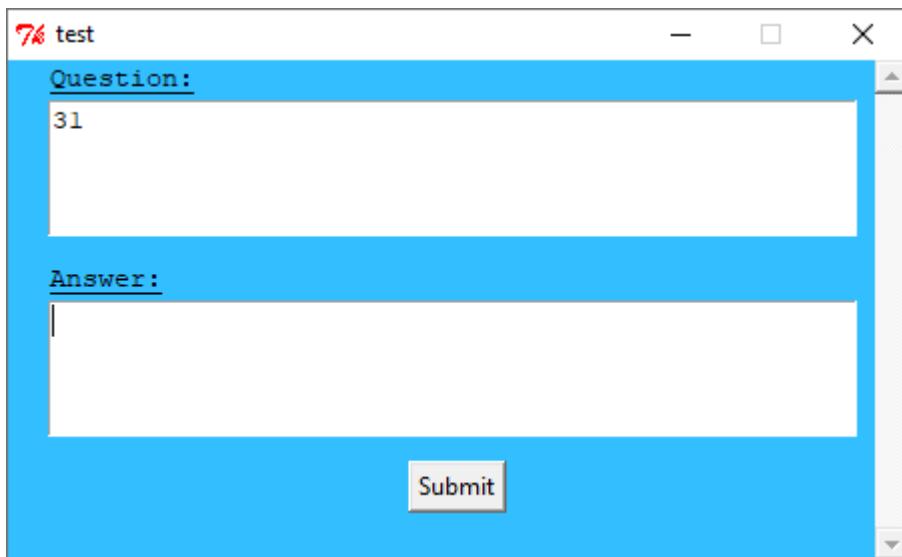
    questionBox.delete(1.0,END)

    i=0
    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            checkAnswer(answer,answers,i)
            if i+1<len(questions):
                #change question variable name
                question=questions[i+1]
                questionBox.insert(END,question)
                break
            #test
        elif i==len(questions):
            print("end of test")
            break

    if answer=="":
        messagebox.showerror("Error","Please enter your answer into the answer box")
        questionBox.insert(END,questions[i])

    questionBox.config(state=DISABLED)
```

However, I receive another error. When, the error message for an empty is closed, it inserts the next question in the test file into the question box. Not only is this not correct, but it also leads to an index error eventually, as the program attempts to insert the value from an index that does not exist in the questions array.



(The first question is “1”, the second question is “3”)

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 148, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers))
  File "N:\GUI.py", line 115, in updateQuestionBox
    questionBox.insert(END,questions[i])
IndexError: list index out of range
```

To fix this I delete the contents of the question box before inserting the question again. This prevents the i value from incrementing (which it does in the first if statement of the while loop as the question box does not equal the current question).

```
i=0
while i!=len(questions):
    if currentQuestion!=questions[i]:
        i=i+1
    elif currentQuestion==questions[i]:
        checkAnswer(answer,answers,i)
        if i+1<len(questions):
            #change question variable name
            question=questions[i+1]
            questionBox.insert(END,question)
            break
    #test
    elif i==len(questions):
        print("end of test")
        break

if answer=="":
    messagebox.showerror("Error","Please enter your answer into the answer box")
    questionBox.delete(1.0,END)
    questionBox.insert(END,questions[i])

questionBox.config(state=DISABLED)
```

Next, I change the question variable's name to newQuestion, to make it easier to differentiate the different question variables. **Otherwise, the purpose of the function is harder to understand, this will make maintaining the program harder**

```

i=0
while i!=len(questions):
    if currentQuestion!=questions[i]:
        i=i+1
    elif currentQuestion==questions[i]:
        checkAnswer(answer,answers,i)
        if i+1<len(questions):
            #change question variable name
            newQuestion=questions[i+1]
            questionBox.insert(END,newQuestion)
            break
        #test
    elif i==len(questions):
        print("end of test")
        break

if answer=="":
    messagebox.showerror("Error","Please enter your answer into the an
questionBox.delete(1.0,END)
questionBox.insert(END,questions[i])

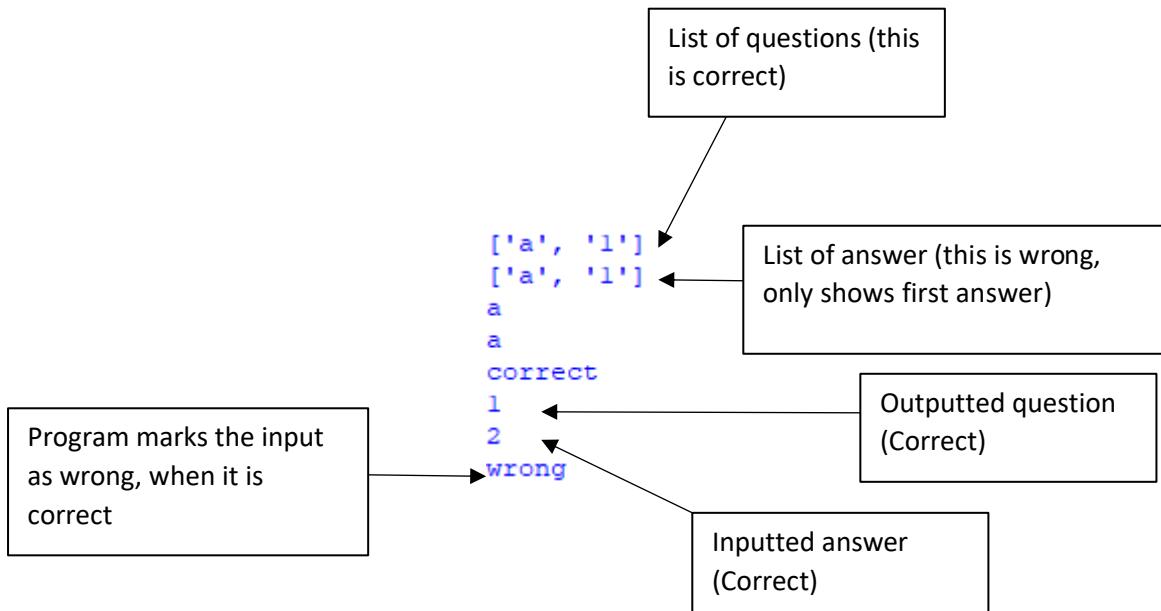
questionBox.config(state=DISABLED)

```

Next, I need to make the program accept multiple correct answers. The program currently only accepts the first answer to a question with multiple answers. **Otherwise, the user may enter a correct answer, but their inputted answer will be marked as incorrect.**

	A	B	C	D
1	Question	Answer		
2	a	a	b	c
3		1	1	2
.				3

(The question '1', has answers of '1','2' and '3'. However, the when the answer '2' is inputted, the program outputs wrong)



To fix this, I will need to access the test file. So, I change the arguments of the `readAllQuestions()` function to pass the name of the test csv file into the necessary function. **Without this, I will not be able to read the questions or answers from the csv file.**

```
def readTestFile(testName, mode):
    testFile=testName+".csv"
    csvFile=open(testFile, 'r+')
    csvFileContent=(csvFile)
    next(csvFile)

    csvList=[]

    for row in csvFileContent:
        csv=row.strip().split(",")
        heading=['Question', 'Answer']
        data=zip(heading,csv)
        DataDict=dict(data)
        csvList.append(DataDict)

    if mode=="Delete":
        if len(csvList)!=0:
            csvList.pop()
            saveTestData(csvList,testFile,DataDict,'w+')
        else:
            messagebox.showerror("File is empty", "There are no more entries")
    elif mode=="Read":
        readAllQuestions(csvList, testName, testFile)
```

## Testing

Then, I make a new test file, using the program, that will make it easier to tell if my program is outputting the correct values.

	A	B	C	D
1	Question	Answer		
2	question1	answer1	answer2	answer3
3	question2	answer4	answer5	answer6
4	question3	answer7	answer8	answer9

Next, I place all the contents of the test file into a list of lists. Then, I need to create a separate list of questions and a list of lists of answers.

I start by creating the list of answers.

```
def readAllQuestions(testList,testName,testFile):
    questions=[]
    answers=[]
    #add better answer function

##    for i in testList:
##        question=i['Question']
##        questions.append(question)

    csvFile=open(testFile)
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

##    print(rows)

    i=1
    j=1
    while i!=len(rows):
        while j!=len(rows[i]):
            answer=rows[i][j]
            answers.append(answer)
            j=j+1
        j=1
        i=i+1

    print(answers)

    testCompletion(testName,testList,questions,answers)
```

(Program output)

```
[['Question', 'Answer'], ['question1', 'answer1', 'answer2', 'answer3'], ['question2', 'answer4', 'answer5', 'answer6'], ['question3', 'answer7', 'answer8', 'answer9']]
Answer
[ 'answer1', 'answer2', 'answer3', 'answer4', 'answer5', 'answer6', 'answer7', 'answer8', 'answer9']
```

## Testing

This successfully creates a list of all answers in a test, next I will need to divide the answers into a list of lists to easily represent all answers to each question separately.

```
def readAllQuestions(testList,testName,testFile):
    questions=[]
    answers=[]
    temp=[]
    #add better answer function

##    for i in testList:
##        question=i['Question']
##        questions.append(question)

    csvFile=open(testFile)
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

##    print(rows)

    i=1
    j=1
    while i!=len(rows):
        while j!=len(rows[i]):
            answer=rows[i][j]
            temp.append(answer)
            j=j+1
        answers.append(temp)
        temp=[]
        j=1
        i=i+1

    print(answers)

    testCompletion(testName,testList,questions,answers)
```

(Program output)

```
[['answer1', 'answer2', 'answer3'], ['answer4', 'answer5', 'answer6'], ['answer7',
', 'answer8', 'answer9']]
```

Next, I will make the list of questions. Without it, there would not be a location to store the questions, which are needed to update the question box to display the next question when the user submits an answer.

```

def readAllQuestions(testList,testName,testFile):
    questions=[]
    answers=[]
    temp=[]
    #add better answer function

##    for i in testList:
##        question=i['Question']
##        questions.append(question)

csvFile=open(testFile)
csvReader=csv.reader(csvFile)
rows=list(csvReader)

##    print(rows)

i=1
while i!=len(rows):
    question=rows[i][0]
    questions.append(question)
    i=i+1

print(questions)

```

## Testing

(Program output)

```
[ 'question1', 'question2', 'question3' ]
```

I will need to modify my checkAnswer() function to use the list of lists instead of a singular list. But, first I remove all unnecessary parameters and arguments. **Otherwise, this makes the functions less likely to cause an error as there are less variables that can be accessed. Also removing redundant variables will make it easier to understand the program, which will make maintaining the program easier.**

ReadAllQuestions function call

```

elif mode=="Read":
    readAllQuestions(csvList,testName,testFile)

elif mode=="Read":
    readAllQuestions(testName,testFile)

```

ReadAllQuestions function definition

```

def readAllQuestions(testList,testName,testFile):
    questions=[]
    answers=[]
    temp=[]

```

```
def readAllQuestions(testName,testFile):
    questions=[]
    answers=[]
    temp=[]
```

TestCompletion function call

```
testCompletion(testName,testList,questions,answers)

testCompletion(testName,questions,answers)
```

TestCompletion function definition

```
def testCompletion(testName,testList,questions,answers):
    completeWindow=Tk()
    completeWindow.geometry("450x250")
    completeWindow.title(testName)
    completeWindow.configure(bg="#33BFFF")

def testCompletion(testName,questions,answers):
    completeWindow=Tk()
    completeWindow.geometry("450x250")
    completeWindow.title(testName)
    completeWindow.configure(bg="#33BFFF")
```

UpdateQuestionBox function call

```
command=lambda: updateQuestionBox(questionBox,questions,answerBox,testList))

,command=lambda: updateQuestionBox(questionBox,questions,answerBox))
```

UpdateQuestionBox function definition

```
def updateQuestionBox(questionBox,questions,answerBox,testList):
    answer=answerBox.get(1.0,'end-1c')

    def updateQuestionBox(questionBox,questions,answerBox):
        answer=answerBox.get(1.0,'end-1c')
```

CheckAnswer function call

```
elif currentQuestion==questions[i]:
    checkAnswer(answer,answers,testList,i,j)
    ...
    ...

elif currentQuestion==questions[i]
    checkAnswer(answer,answers,i,j)
```

CheckAnswer function definition

```
def checkAnswer(answer,answers,testList,i,j):

def checkAnswer(answer,answers,i,j):
    ...
    ...
    ...
```

Then, I modify the checkAnswer and updateQuestionBox function so that they are compatible with the new list. **Otherwise, the program would not be able to read the contents of the list as it is now a 2d list.** However, when the program is run an error appears.

```

def updateQuestionBox(questionBox, questions, answerBox):
    answer=answerBox.get(1.0, 'end-1c')
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0, 'end-1c')

    questionBox.delete(1.0,END)

    i=0
    j=0
    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            checkAnswer(answer,answers,i,j)

def checkAnswer(answer,answers,i,j):
    answersForCurrentQuestion=answers[i]

    while j!=len(answersForCurrentQuestion):
        if answer==answersForCurrentQuestion[j]:
            print("correct")
            break
        elif answer!=answersForCurrentQuestion[j]:
            print("wrong")

```

## Testing

(Program output)

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 165, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuest
ionBox(questionBox,questions,answerBox))
  File "N:\GUI.py", line 119, in updateQuestionBox
    checkAnswer(answer,answers,i,j)
NameError: global name 'answers' is not defined

```

To fix this I pass the answers array into all functions up to the checkAnswer function call. **Otherwise, the checkAnswer function cannot access the answers array, therefore it cannot check if the inputted answer is correct by matching it to the answers for the question in the array.**

```

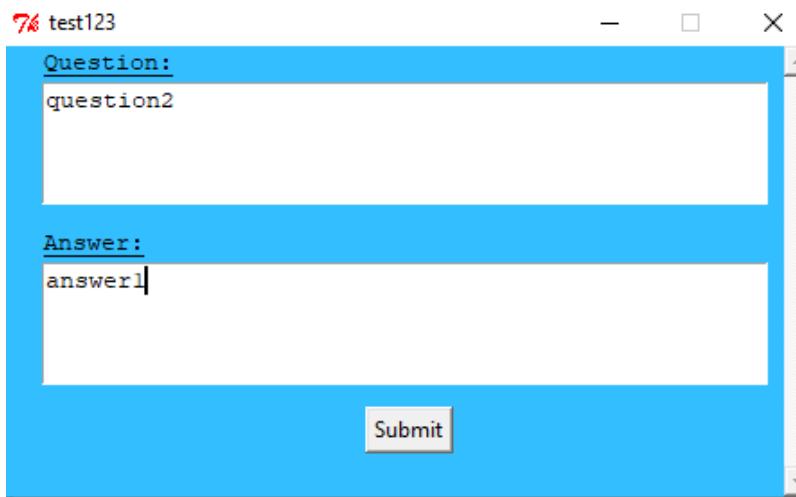
.command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers))

def updateQuestionBox(questionBox,questions,answerBox,answers):

```

## Testing

The program correctly identifies a correct answer if it is the first answer in the array, if it is not the first answer in the array, then it will infinitely output “wrong” until the program crashes.



(Program output)

```
['question1', 'question2', 'question3']
[['answer1', 'answer2', 'answer3'], ['answer4', 'answer5', 'answer6'], ['answer7',
  'answer8', 'answer9']]
correct
```



(Program output)

```
wrong
```

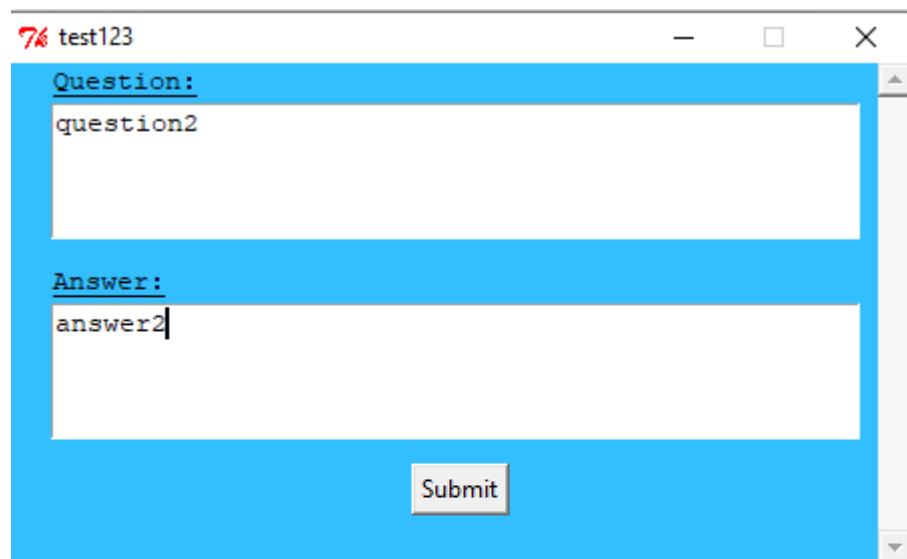
I add a line of code to increment the value of j. Without this, the program will forever enter the loop as the value of j will never match the length of the list, therefore if the user were to input a wrong answer they would not be able to finish the test.

```
def checkAnswer(answer,answers,i,j):
    answersForCurrentQuestion=answers[i]

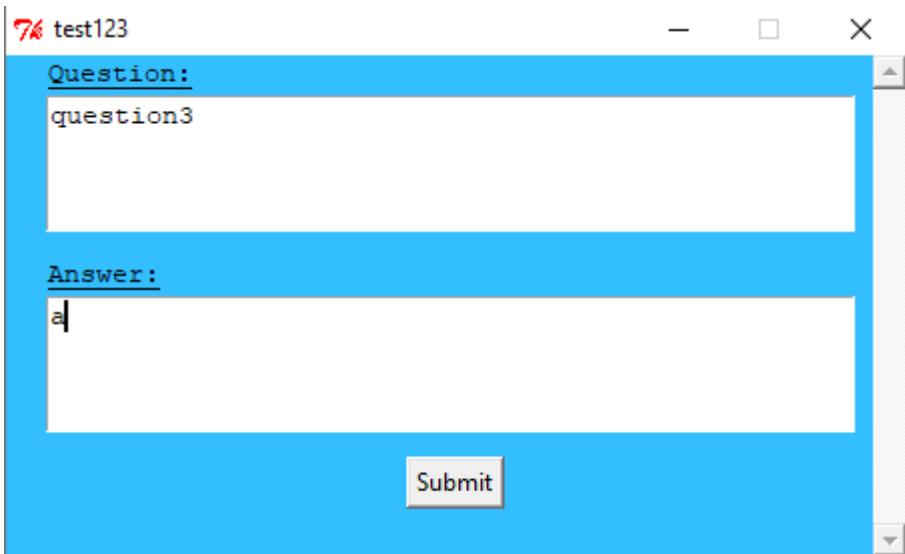
    while j!=len(answersForCurrentQuestion):
        if answer==answersForCurrentQuestion[j]:
            print("correct")
            break
        elif answer!=answersForCurrentQuestion[j]:
            print("wrong")
            j=j+1
```

## Testing

Now, the program correctly exits the loop if either the answer is right or wrong.



```
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.150
32
Type "copyright", "credits" or "license()" for more info
>>> ===== RESTART =====
>>>
['question1', 'question2', 'question3']
[['answer1', 'answer2', 'answer3'], ['answer4', 'answer5
', 'answer8', 'answer9']]
wrong
correct
```



(Program output)

```
wrong  
wrong  
wrong  
|
```

### Marks system function

Next, I need to implement a score system. To do this I create a marks variable and pass it through the necessary functions. **Without this, the program has no location to store the marks achieved in a test.**

```
marks=0  
i=0  
j=0  
while i!=len(questions):  
    if currentQuestion!=questions[i]:  
        i=i+1  
    elif currentQuestion==questions[i]:  
        checkAnswer(answer,answers,i,j,marks)  
        if i+1<len(questions):  
            newQuestion=questions[i+1]  
            questionBox.insert(END,newQuestion)  
            break  
    #test  
    elif i==len(questions):  
        print("end of test")  
    break
```

```

def checkAnswer(answer, answers, i, j, marks):
    answersForCurrentQuestion=answers[i]

    while j!=len(answersForCurrentQuestion):
        if answer==answersForCurrentQuestion[j]:
            print("correct")
            marks=marks+1
            break
        elif answer!=answersForCurrentQuestion[j]:
            print("wrong")
            j=j+1
    print(marks)

```

## Testing

The program runs with no errors. However, the marks variable resets each time a new question appears  
(Program output)

```

['question1', 'question2', 'question3']
[['answer1', 'answer2', 'answer3'], ['answer4', 'answer5', 'answer6'], ['answer7',
', 'answer8', 'answer9']]
correct
1
correct
1
wrong
wrong
wrong
0

```

I can fix this while implementing the next feature, which is to save the marks of a user to a csv. Each time the user gets a question correct the user's marks will increase in the csv file.

First, I add to my gradeFileCreator function. I use this function to also create another csv that will be used to store the marks. **Otherwise, I will need to create a separate function to create the marks csv file, this would make the program more complicated and more difficult to maintain in the future. By adding onto this already created function, I can save time both coding the creation of the marks file and save time in the future if I need to make modifications to my code.**

```

def gradeFileCreator(testName,A,B,C,D,E,F,gradeWindow):
    rootFile=os.path.abspath(os.curdir)
    gradeFileName=testName.strip(".csv")
    dst1File=gradeFileName+" - grades"+".csv"
    dst2File=gradeFileName+" - marks"+".csv"
    rootFile=rootFile+"\empty.csv"

    shutil.copy2(rootFile,dst1File)
    shutil.copy2(rootFile,dst2File)

    openGradeFile(dst1File,A,B,C,D,E,F,gradeWindow)

```

The newly added code succeeds in making a new csv file and naming it appropriately based on the user's input. **Without this code, the “testname - marks.csv” file cannot be created, therefore a user's results after completing a test cannot be saved for evaluating by a teacher.**

newTestFile	15/03/2023 12:11	Microsoft Excel C...	1 KB
newTestFile - grades	15/03/2023 12:11	Microsoft Excel C...	1 KB
newTestFile - marks	06/12/2022 14:57	Microsoft Excel C...	1 KB

Then I make a copy of my openGradeFile function to add appropriate headings to the csv

```
def openMarkFile(file):
    csvFile=open(file,'r+')
    csvFile.write('Username,Marks,Grades\n')
    fileRead=csvFile.read()

    for item in fileRead:
        csvFile.write('{Username},{Marks},{Grades}'.format(**item))
        csvFile.write('\n')
    csvFile.close()
```

	A	B	C
1	Username	Marks	Grades
2			
3			
4			
5			
6			
7			
8			
9			
..			

Next, I will create a function that executes when someone starts completing a test and adds their test details as they complete the test. In order to do this, I need to pass the username variable from the loginAccount function through all functions up until the readAllQuestions function. **Otherwise, the username cannot be accessed when completing a test, therefore the test results cannot be linked to a user.**

LoginAccount function to secondaryMenu function

```
'def secondaryMenu(username,adminStatus):
    -----
if loginStatus==True:
    secondaryMenu(username,adminStatus)
.'
```

SecondaryMenu function to nameTestFile function

```
def nameTestFile(title,function,username):
    command=lambda: nameTestFile("Complete a test","Complete",username)
    command=lambda: nameTestFile("Create a test","Create",username)
```

NameTestFile function to checkFileName function

```
def checkFileName(testName,username):
    ,command=lambda:checkFileName(nameOfTextBox,username))
```

CheckFileName function to readTestFile function

```
def readTestFile(testName, mode, username):
    testFileName = testName + ".csv"
    if os.path.isfile(file)==True:
        readTestFile(testName, "Read", username)
    else:
        print("File does not exist")
```

ReadTestFile function to ReadAllQuestions function

```
def readAllQuestions(testName, testFile, username):
    elif mode=="Read":
        readAllQuestions(testName, testFile, username)
```

Now, the username variable can be accessed during the test completion process

Next, I add the testName variable to the arguments and parameters of the updateQuestionBox function, so that the test's marks file can be found. **Without this, the marks file cannot be accessed as the test name variable needs to be passed to the answer function to open the csv file.**

```
updateQuestionBox(questionBox, questions, answerBox, answers, marks, testName)
```

```
def updateQuestionBox(questionBox, questions, answerBox, answers, marks, testName):
    for i in range(len(questions)):
        questionBox.insert(END, questions[i])
```

I also pass the testName variable into the next function, which is the checkAnswer function, for the same reason.

```
while i!=len(questions):
    if currentQuestion!=questions[i]:
        i=i+1
    elif currentQuestion==questions[i]:
        checkAnswer(answer, answers, i, j, marks, testName)
        if i+1<len(questions):
            newQuestion=questions[i+1]
            questionBox.insert(END, newQuestion)
            break
    #test
    elif i==len(questions):
        print("end of test")
        break

def checkAnswer(answer, answers, i, j, marks, testName):
    answersForCurrentQuestion=answers[i]

    while j!=len(answersForCurrentQuestion):
        if answer==answersForCurrentQuestion[j]:
```

## Testing

### Error

I modify the checkAnswer function to open the marks file and when the marks file only contains the default headings there is no error. However, when testing I realise that as soon as the file has data it will cause an error.

```
def checkAnswer(answer,answers,i,j,marks,testName)
    file=testName+" - marks.csv"
    openMarkFile(file)
```

	A	B	C
1	Username	Marks	Grades
2	Example1	1	A
3	Example2	2	B
4	Example3	3	C

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 197, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName))
  File "N:\GUI.py", line 148, in updateQuestionBox
    checkAnswer(answer,answers,i,j,marks,testName)
  File "N:\GUI.py", line 114, in checkAnswer
    openMarkFile(file)
  File "N:\GUI.py", line 260, in openMarkFile
    csvFile.write('{Username},{Marks},{Grades}'.format(**item))
TypeError: format() argument after ** must be a mapping, not str
```

A type error is received when there is data other than the headings in the file. I realise that this is likely an issue due to the headings as, even if the headings have already been written the program will attempt to write them again. To fix this I add an if statement to check whether the csv file is empty. If it is empty, it will write the headings, if it is not it will not do anything for now. **Without this, the program will continue to attempt to write headings every time the file is opened, since the file won't be in the specific format necessary to write the headings, when the headings are already written, a TypeError will be produced.**

```
def openMarkFile(file):
    csvFile=open(file,'r+')
    fileRead=csvFile.read()
    if fileRead.strip()=='':
        csvFile.write('Username,Marks,Grades\n')
        fileRead=csvFile.read()

        for item in fileRead:
            csvFile.write('{Username},{Marks},{Grades}'.format(**item))
            csvFile.write('\n')
        csvFile.close()
        print("headings have been written")
    else:
        print("headings are already written")
```

## Testing

I test this if statement by modifying the marks file accordingly and checking the program output.

When the file has data, the program does not attempt to write headings

	A	B	C
1	Username	Marks	Grades
2	Example1		1 A
3	Example2		2 B
4	Example3		3 C

```
['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings are already written
headings are already written
```

When the file has no data, adds headings but to the wrong row

	A	B	C
1			
2			
3			
4			

```
>>>
['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings have been written
```

	A	B	C
1			
2	Username	Marks	Grades
3			
4			
5			

To fix this problem, I open the file in write mode when adding the headings. This overwrites all contents of the csv file, ensuring that all writing is added to the first row. I also move the read function before this to prevent an error. **Otherwise, the program will attempt to read from the file when it is in write mode, which causes an error.**

	A	B	C
1	Username	Marks	Grades
2			
3			
4			

```
>>>
['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings have been written
```

```
def openMarkFile(file):
    csvFile=open(file,'r+')
    fileRead=csvFile.read()
    if fileRead.strip()=='':
        fileRead=csvFile.read()
        csvFile=open(file,'w+')
        csvFile.write('Username,Marks,Grades\n')

        for item in fileRead:
            csvFile.write('{Username},{Marks},{Grades}'.format(**item))
            csvFile.write('\n')
        csvFile.close()

        print("headings have been written")
    else:
        print("headings are already written")
```

Next, I will create a function to set up the mark file with the username and some default data, this will be needed to increment the mark whenever a correct answer is given. This function will be called just before the GUI is created.

```
def setMarkFile(username,testName):
    file=testName+" - marks.csv"
    csvFile=open(file,'a')
    newLine=username+",0,F\n"
    csvFile.write(newLine)
    csvFile.close()
```

```

def readAllQuestions(testName,testFile,username):
    questions=[]
    answers=[]
    temp=[]

    csvFile=open(testFile)
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    i=1
    while i!=len(rows):
        question=rows[i][0]
        questions.append(question)
        i=i+1

    print(questions)

    i=1
    j=1
    while i!=len(rows):
        while j!=len(rows[i]):
            answer=rows[i][j]
            temp.append(answer)
            j=j+1
        answers.append(temp)
        temp=[]
        j=1
        i=i+1

    print(answers)

setMarkFile(username,testName)

testCompletion(testName,questions,answers)

```

## Testing

(username of account used to login in is 'a')

	A	B	C
1	Username	Marks	Grades
2	a	0	F
3			
4			

Next, I will now modify the checkAnswer function to increment the marks each time an answer is correct.

Initially, the openMarkFile function is called during the checkAnswer function to ensure that the csv file is in the correct format, however it would be unnecessary to call it in the checkAnswer function as it only needs to be checked once while the checkAnswer function is called several times. Therefore, I move the openMarkFile to the readAllQuestions function.

```

def checkAnswer(answer, answers, i, j, marks, testName):
    file=testName+" - marks.csv"
    openMarkFile(file)

def checkAnswer(answer, answers, i, j, marks, testName):
    #temp statement to prevent error
    return(answer)

(readAllQuestions function)

    i=1
    j=1
    while i!=len(rows):
        while j!=len(rows[i]):
            answer=rows[i][j]
            temp.append(answer)
            j=j+1
        answers.append(temp)
        temp=[]
        j=1
        i=i+1

    print(answers)

    setMarkFile(username,testName)

    openMarkFile(testName+' - marks.csv')

    testCompletion(testName,questions,answers)

```

First, I try to organise the csv file's contents into a list of lists.

```

# Checks the inputted answer against the actual answers
def checkAnswer(answer, answers, i, j, marks, testName):
    file=testName+" - marks.csv"
    csvFile=open(file,'r+')

    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    print(rows)

[['Username', 'Marks', 'Grades'], ['a', '0', 'F'], ['a', '0', 'F'], ['a', '0', 'F'], ['a', '0', 'F']]

```

Next, I try to find the last recorded entry by a user, this will be the entry to modify. By doing this, it allows a user to attempt a test multiple times and only modify their most recent entry in the marks file.

## Testing

```

def checkAnswer(answer,answers,i,j,marks,testName,username):
    file=testName+" - marks.csv"
    csvFile=open(file,'r+')

    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    print(rows)

    lastAttemptByUser=0

    for i in range(len(rows)-1,1,-1):
        if rows[i][0]==username:
            lastAttemptByUser=i
            print(lastAttemptByUser)
            break

```

	A	B	C
1	Username	Marks	Grades
2	a	0	F
3	a	0	F
4	a	0	F
5	a	0	F
6	a	0	F
7	a	0	F
8	a	0	F
9			

```

['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '0', 'F'], ['a', '0', 'F'], ['a', '0', 'F'],
 ['a', '0', 'F'], ['a', '0', 'F'], ['a', '0', 'F'], ['a', '0', 'F']]
7

```

This works perfectly, but I realise this should also only needs to be executed once like the openMarkFile call. So I move the new code to the setMarkFile function.

(I also add a close statement at the end, to prevent any unwanted modifications to the file)

```
def setMarkFile(username,testName):
    file=testName+" - marks.csv"
    csvFile=open(file,'a')
    newLine=username+",0,F\n"
    csvFile.write(newLine)
    csvFile.close()

    file=testName+" - marks.csv"
    csvFile=open(file,'r+')

    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    print(rows)

    lastAttemptByUser=0

    for i in range(len(rows)-1,1,-1):
        if rows[i][0]==username:
            lastAttemptByUser=i
            print(lastAttemptByUser)
            break

    csvFile.close()
```

Next, I realise that the function calls in my `readAllQuestions` function is in the wrong order. So, I rearrange it. **Otherwise, if the headers for the marks file have not been added it will set up the mark file with a new test attempt entry then check if the headings are present, which it will state that it is present as the file is not empty.**

(Before)

```
setMarkFile(username,testName)

openMarkFile(testName+' - marks.csv')

testCompletion(testName,questions,answers,username)
```

(After)

```
openMarkFile(testName+' - marks.csv')

setMarkFile(username,testName)

testCompletion(testName,questions,answers,username)
```

I also realise I need to pass a variable from `setMarkFile` to `testCompletion`. So, I move the `testCompletion` function call to the `setMarkFile` and move all necessary arguments into the `setMarkFile`

(`readAllQuestions` function)

```

openMarkFile(testName+' - marks.csv')

setMarkFile(username,testName,questions,answers)|

(setMarkFile function)

    for i in range(len(rows)-1,1,-1):
        if rows[i][0]==username:
            lastAttemptByUser=i
            print(lastAttemptByUser)
            break

    csvFile.close()

    testCompletion(testName,questions,answers,username)

```

## Error

I receive a type error.

```

['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings are already written
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 532, in <lambda>
    submitButton=Button(fileCreator,text="Submit",command=lambda:checkFileName(n
ameOfTextBox,username))
  File "N:\GUI.py", line 58, in checkFileName
    readTestFile(testName,"Read",username)
  File "N:\GUI.py", line 48, in readTestFile
    readAllQuestions(testName,testFile,username)
| File "N:\GUI.py", line 124, in readAllQuestions
  setMarkFile(username,testName,questions,answers)
TypeError: setMarkFile() takes exactly 2 positional arguments (4 given)

```

I fix this by adding the new parameters.

```

def setMarkFile(username,testName,questions,answers):
    file=testName+" - marks.csv"
    csvFile=open(file,'a')
    newLine=username+",0,F\n"
    csvFile.write(newLine)|
    csvFile.close()

```

Next, I fix the parameters and arguments of the checkAnswer function as there are a lot of variables I no longer need. I also pass the new lastAttemptByUser and rows variables

```
for i in range(len(rows)-1,1,-1):
    if rows[i][0]==username:
        lastAttemptByUser=i
        print(lastAttemptByUser)
        break

csvFile.close()

testCompletion(testName,questions,answers,username,lastAttemptByUser,rows)

def testCompletion(testName,questions,answers,username,rowOfEntry,rows):
    marks=0

    completeWindow=Tk()
    completeWindow.geometry("450x250")
    completeWindow.title(testName)
    completeWindow.configure(bg="#33BFFF")

    completeWindow.resizable(False,False)

command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName,username,rowOfEntry,rows)

def updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName,username,rowOfEntry,rows):
    #delete answer box when next question is added (prototype 2)
    answer=answerBox.get(1.0,'end-lc')
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0,'end-lc')

    questionBox.delete(1.0,END)

    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows)
            if i+1<len(questions):
                newQuestion=questions[i+1]
                questionBox.insert(END,newQuestion)
                break
            #test
        elif i==len(questions):
            print("end of test")
            break

    # Checks the inputted answer against the actual answers
def checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows):
    file=testName+" - marks.csv"
    csvFile=open(file,'r+')
```

Finally, I will add the code to increment the marks if a correct answer is entered.

## Error

I create the following code but receive an error when the function executes.

```

# Checks the inputted answer against the actual answers
def checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows):
    file=testName+" - marks.csv"
    csvFile=open(file,'w')

    correctAnswers=answers[i]

    while j!=len(correctAnswers):
        if answer==correctAnswers[j]:
            rows[rowOfEntry][1]=rows[rowOfEntry][1]+1
            csvFile.write(rows)
            print("correct")
            break
        elif answer!=correctAnswers[j]:
            print("wrong")
            j=j+1

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 209, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName,username,rowOfEntry,rows))
  File "N:\GUI.py", line 160, in updateQuestionBox
    checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows)
  File "N:\GUI.py", line 136, in checkAnswer
    rows[rowOfEntry][1]=rows[rowOfEntry][1]+1
TypeError: Can't convert 'int' object to str implicitly

```

## Testing

Also, the contents of the marks csv file is deleted. This is because the file is opened in 'write' mode, which overwrites the current contents of the file, since the error prevents the program from rewriting to the csv file, the file becomes empty

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

## Error

I try to fix the error by converting the value in the csv file (the mark) to an integer but receive a different error.

```

while j!=len(correctAnswers):
    if answer==correctAnswers[j]:
        rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
        csvFile.write(rows)
        print("correct")
        break
    elif answer!=correctAnswers[j]:
        print("wrong")
        j=j+1

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 209, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName,username,rowOfEntry,rows))
  File "N:\GUI.py", line 160, in updateQuestionBox
    checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows)
  File "N:\GUI.py", line 136, in checkAnswer
    rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
ValueError: invalid literal for int() with base 10: 'Marks'

```

After adding a line of code to check the value of rowOfEntry along with the previous error message. I can tell that the rowOfEntry variable is set to 0 and does not change

```

def checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows):
    file=testName+" - marks.csv"
    csvFile=open(file,'w')

    correctAnswers=answers[i]

    print(rowOfEntry)

['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings have been written
[['Username', 'Marks', 'Grades'], ['a', '0', 'F']]
0 ←
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 211, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName,username,rowOfEntry,rows))
  File "N:\GUI.py", line 162, in updateQuestionBox
    checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows)
  File "N:\GUI.py", line 138, in checkAnswer
    rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
ValueError: invalid literal for int() with base 10: 'Marks'

```

Value of rowOfEntry

I also realise that a for loop is not executing as it does not output the print statement defined. Therefore, I know I need to fix the range on the loop.

```

def setMarkFile(username, testName, questions, answers):
    file=testName+" - marks.csv"
    csvFile=open(file, 'a')
    newLine=username+",0,F\n"
    csvFile.write(newLine)
    csvFile.close()

    file=testName+" - marks.csv"
    csvFile=open(file, 'r+')

    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    print(rows)

lastAttemptByUser=0

for i in range(len(rows)-1,1,-1):
    if rows[i][0]==username:
        lastAttemptByUser=i
        print(lastAttemptByUser)
        break

```

Print statement is not outputting, so the loop is not being entered or the if statement is never met

## Error

I add define the start of the range as a variable, which seems to fix the error, but a new error appears.

```

lastAttemptByUser=0

start=len(rows)-1
print(start)

for i in range(start,1,-1):
    print(i)
    if rows[i][0]==username:
        lastAttemptByUser=i
        print(lastAttemptByUser)
        break

```

```
['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '0', 'F'], ['a', '0', 'F']]
2
2
2
2
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 216, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName,username,rowOfEntry,rows))
  File "N:\GUI.py", line 167, in updateQuestionBox
    checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows)
  File "N:\GUI.py", line 144, in checkAnswer
    csvFile.write(rows)
TypeError: must be str, not list
```

Using this code, I realise that the `TypeError` is given when there is more than 1 user in the csv file and the `ValueError` is given when there is only 1 user in the csv file. It looks like when there is 1 user in the csv file, the program will take row 0 as the row where the user is stored, however this row is the heading which leads to a value error as the row 0 contains the heading 'Marks', whereas all other rows below this, row 1 onwards, will contain an actual number value.

```
lastAttemptByUser=0

print(rows)

for i in range(len(rows)-1,1,-1):
    if rows[i][0]==username:
        lastAttemptByUser=i
        break

print(lastAttemptByUser)
csvFile.close()

testCompletion(testName,questions,answers,username,lastAttemptByUser,rows)
```

```

>>>
['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '0', 'F'], ['a', '0', 'F']]
2
2
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 212, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName,username,rowOfEntry,rows))
  File "N:\GUI.py", line 163, in updateQuestionBox
    checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows)
  File "N:\GUI.py", line 140, in checkAnswer
    csvFile.write(rows)
TypeError: must be str, not list
>>> ===== RESTART =====

```

More than 1 user in the csv file leads to a TypeError

```

>>>
['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings have been written
[['Username', 'Marks', 'Grades'], ['a', '0', 'F']]
0
0
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 212, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName,username,rowOfEntry,rows))
  File "N:\GUI.py", line 163, in updateQuestionBox
    checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows)
  File "N:\GUI.py", line 137, in checkAnswer
    rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
ValueError: invalid literal for int() with base 10: 'Marks'

```

Only 1 user in the csv file leads to a ValueError

To fix this I define `lastAttemptByUser` as 1, this causes a `TypeError` when there is only 1 user in the csv file, which means that this function is fixed and I can now move on to fixing the `TypeError`, which is being caused by the `checkAnswer` function.

```

lastAttemptByUser=1

print(rows)

for i in range(len(rows)-1,1,-1):
    if rows[i][0]==username:
        lastAttemptByUser=i
        break

print(lastAttemptByUser)
csvFile.close()

testCompletion(testName,questions,answers,username,lastAttemptByUser,rows)

```

```

>>>
['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings have been written
[['Username', 'Marks', 'Grades'], ['a', '0', 'F']]
1
1
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 212, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName,username,rowOfEntry,rows))
  File "N:\GUI.py", line 163, in updateQuestionBox
    checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows)
  File "N:\GUI.py", line 140, in checkAnswer
    csvFile.write(rows)
TypeError: must be str, not list

```

I fix this by writing the list into the csv file in a format that is suitable for csv. However, the rows below the heading have a 1 line gap.

```

# Checks the inputted answer against the actual answers
def checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows):
    file=testName+" - marks.csv"
    csvFile=open(file,'w')
    csvWriter=csv.writer(csvFile)

    correctAnswers=answers[i]

    print(rowOfEntry)

    while j!=len(correctAnswers):
        if answer==correctAnswers[j]:
            rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
            csvWriter.writerows(rows)
            print("correct")
            break
        elif answer!=correctAnswers[j]:
            print("wrong")
            j=j+1

```

For testing

Username	Marks	Grades
a	2	F

## Testing

I quickly test if this is a problem due to only 1 user being registered by manually adding some users

	A	B	C
1	Username	Marks	Grades
2	test	10	A
3	a	2	F
4	b	0	F
5	a	99	B

The program runs with no errors, but I realise a gap is added between all entries

	A	B	C
1	Username	Marks	Grades
2			
3	test	10	A
4			
5	a	2	F
6			
7	b	0	F
8			
9	a	99	B
10			
11	a	2	F

I reset the table to the original test data and modify the parameters of opening the csv file, by specifying the newline terminator, python will not add its own new line. **Without specifying the new line terminator, python will add its own new line, after each row is written, which causes an additional new line**

	A	B	C
1	Username	Marks	Grades
2	test	10	A
3	a	2	F
4	b	0	F
5	a	99	B

```
def checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows):
    file=testName+" - marks.csv"
    csvFile=open(file,'w',newline='')
    csvWriter=csv.writer(csvFile)

    correctAnswers=answers[i]

    print(rowOfEntry)

    while j!=len(correctAnswers):
        if answer==correctAnswers[j]:
            rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
            csvWriter.writerows(rows)
            print("correct")
            break
        elif answer!=correctAnswers[j]:
            print("wrong")
            j=j+1
```

	A	B	C
1	Username	Marks	Grades
2	test	10	A
3	a	2	F
4	b	0	F
5	a	99	B
6	a	2	F

This function is now completed, next I remove the username parameter and argument from all functions that do not need it. **Without removing this, the username variable could be unintentionally modified leading to errors or incorrect data being saved on the csv file.**

```

# Adds new user test attempt to the marks file
def setMarkFile(username,testName,questions,answers):
    file=testName+" - marks.csv"
    csvFile=open(file,'a')
    newLine=username+",0,F\n"
    csvFile.write(newLine)
    csvFile.close()

    file=testName+" - marks.csv"
    csvFile=open(file,'r+')

    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    lastAttemptByUser=1

    print(rows)

    for i in range(len(rows)-1,1,-1):
        if rows[i][0]==username:
            lastAttemptByUser=i
            break

    print(lastAttemptByUser)
    csvFile.close()

    testCompletion(testName,questions,answers,username,lastAttemptByUser,rows)

```

```
testCompletion(testName,questions,answers,lastAttemptByUser,rows)
```

I also realise that the marks variable is not needed so I will also be removing this as well to prevent any unintended effects.

```

# GUI for test completion
def testCompletion(testName,questions,answers,username,rowOfEntry,rows):
    marks=0
    | 
    completeWindow=Tk()
    completeWindow.geometry("450x250")
    completeWindow.title(testName)
    completeWindow.configure(bg="#33BFFF")

    completeWindow.resizable(False,False)

    scrollbar=Scrollbar(completeWindow)
    scrollbar.pack(side=RIGHT,fill=Y)

    questionLabel=Label(completeWindow,text="Question:",bg="#33BFFF",font=("Cour
    questionLabel.place(x=20,y=0)

    questionBox=Text(completeWindow,width=50,height=4,wrap=WORD)
    questionBox.place(x=20,y=20)

    questionBox.insert(END,questions[0])

    questionBox.config(yscrollcommand=scrollbar.set,state=DISABLED)
    scrollbar.config(command=questionBox.yview)

```

```
def testCompletion(testName,questions,answers,rowOfEntry,rows):  
  
    completeWindow=Tk()  
    completeWindow.geometry("450x250")  
    completeWindow.title(testName)  
    completeWindow.configure(bg="#33BFFF")  
  
    completeWindow.resizable(False,False)  
  
    scrollbar=Scrollbar(completeWindow)  
    scrollbar.pack(side=RIGHT,fill=Y)  
  
    questionLabel=Label(completeWindow,text="Question:",bg="#33BFFF",font=("Cour  
questionLabel.place(x=20,y=0)  
  
(function call in testCompletion function)  
  
updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName,username,rowOfEntry,rows))  
updateQuestionBox(questionBox,questions,answerBox,answers,testName,rowOfEntry,rows))  
  
# Updates the question box after an answer has been submitted  
def updateQuestionBox(questionBox,questions,answerBox,answers,marks,testName,username,rowOfEntry,rows):  
    #delete answer box when next question is added (prototype 2)  
    answer=answerBox.get(1.0,'end-1c')  
    questionBox.config(state=NORMAL)  
  
# Updates the question box after an answer has been submitted  
def updateQuestionBox(questionBox,questions,answerBox,answers,testName,rowOfEntry,rows):  
    #delete answer box when next question is added (prototype 2)  
    answer=answerBox.get(1.0,'end-1c')  
    questionBox.config(state=NORMAL)  
  
(function call in updateQuestionBox function)  
  
i=0  
j=0  
while i!=len(questions):  
    if currentQuestion!=questions[i]:  
        i=i+1  
    elif currentQuestion==questions[i]:  
        checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows)  
        if i+1<len(questions):  
            newQuestion=questions[i+1]  
            questionBox.insert(END,newQuestion)  
        break
```

```
i=0
j=0
while i!=len(questions):
    if currentQuestion!=questions[i]:
        i=i+1
    elif currentQuestion==questions[i]:
        checkAnswer(answer,answers,i,j,testName,rowOfEntry,rows)
        if i+1<len(questions):
            newQuestion=questions[i+1]
            questionBox.insert(END,newQuestion)
            break

# Checks the inputted answer against the actual answers
def checkAnswer(answer,answers,i,j,marks,testName,username,rowOfEntry,rows):

# Checks the inputted answer against the actual answers
def checkAnswer(answer,answers,i,j,testName,username,rowOfEntry,rows):
```

I also add a close statement to the checkAnswer function. Otherwise, the file may be unintentionally modified by another part of the program.

```
# Checks the inputted answer against the actual answers
def checkAnswer(answer,answers,i,j,testName,rowOfEntry,rows):

##    pass questions into this function and execute following code
##    create results GUI and execute if statement is met
##        if i==len(questions):
##            print("end of test")

file=testName+" - marks.csv"
csvFile=open(file,'w',newline='')
csvWriter=csv.writer(csvFile)

correctAnswers=answers[i]

print(rowOfEntry)

while j!=len(correctAnswers):
    if answer==correctAnswers[j]:
        rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
        csvWriter.writerows(rows)
        print("correct")
        break
    elif answer!=correctAnswers[j]:
        print("wrong")
        j=j+1

csvFile.close()
```

I also remove the variable `j` from the `updateQuestionBox` function and instead define `j` in the `checkAnswer` function. **Otherwise, the variable may be modified unintentionally, which can lead to an error.**

```
# Updates the question box after an answer has been submitted
def updateQuestionBox(questionBox,questions,answerBox,answers,testNa
    #delete answer box when next question is added (prototype 2)
    answer=answerBox.get(1.0,'end-1c')
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0,'end-1c')

    questionBox.delete(1.0,END)

    i=0
    j=0

    if i==len(questions):
        completeWindow.destroy()
        print("test")

    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            checkAnswer(answer,answers,i,j,testName,rowOfEntry,rows)
            if i+1<len(questions):
                newQuestion=questions[i+1]
                questionBox.insert(END,newQuestion)
            break|
```

```
# Updates the question box after an answer has been submitted
def updateQuestionBox(questionBox,questions,answerBox,answers,test
    #delete answer box when next question is added (prototype 2)
    answer=answerBox.get(1.0,'end-1c')
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0,'end-1c')

    questionBox.delete(1.0,END)

    i=0

    if i==len(questions):
        completeWindow.destroy()
        print("test")

    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            checkAnswer(answer,answers,i,testName,rowOfEntry,rows)
            if i+1<len(questions):
                newQuestion=questions[i+1]
                questionBox.insert(END,newQuestion)
            break
```

```
# Checks the inputted answer against the actual answers
def checkAnswer(answer,answers,i,j,testName,rowOfEntry,rows):

##    pass questions into this function and execute following code
##    create results GUI and execute if statement is met
##        if i==len(questions):
##            print("end of test")

file=testName+" - marks.csv"
csvFile=open(file,'w',newline='')
csvWriter=csv.writer(csvFile)

correctAnswers=answers[i]

print(rowOfEntry)

while j!=len(correctAnswers):
    if answer==correctAnswers[j]:
        rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
        csvWriter.writerows(rows)
        print("correct")
        break
    elif answer!=correctAnswers[j]:
        print("wrong")
        j=j+1

csvFile.close()
```

```
# Checks the inputted answer against the actual answers
def checkAnswer(answer,answers,i,testName,rowOfEntry,rows):

    ##      pass questions into this function and execute following
    ##      create results GUI and execute if statement is met
    ##          if i==len(questions):
    ##              print("end of test")

    file=testName+" - marks.csv"
    csvFile=open(file,'w',newline='')
    csvWriter=csv.writer(csvFile)

    correctAnswers=answers[i]

    print(rowOfEntry)

    j=0

    while j!=len(correctAnswers):
        if answer==correctAnswers[j]:
            rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
            csvWriter.writerow(rows)
            print("correct")
            break
        elif answer!=correctAnswers[j]:
            print("wrong")
            j=j+1

    csvFile.close()
```

### Test results summary system

Next, I will add a results screen that appears after the last question has been answered.

Before this, I need to create a function to calculate the grade based on the user's marks and the grade boundaries for the test. In order to make this I need to find where the test stops, I already have a statement that should execute at the end of a test, however no output is given

```

----- -----
def updateQuestionBox(questionBox,questions,answerBox,answers,test1
    #delete answer box when next question is added (prototype 2)
    answer=answerBox.get(1.0,'end-1c')
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0,'end-1c')

    questionBox.delete(1.0,END)

    i=0

    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            checkAnswer(answer,answers,i,testName,rowOfEntry,rows)
            if i+1<len(questions):
                newQuestion=questions[i+1]
                questionBox.insert(END,newQuestion)
                break
            #test
            elif i==len(questions):
                print("end of test")
                break
        if answer=="":

            messagebox.showerror("Error","Please enter your answer into the answer box")
            questionBox.delete(1.0,END)
            questionBox.insert(END,questions[i])

        questionBox.config(state=DISABLED)

['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings have been written
[['Username', 'Marks', 'Grades'], ['a', '0', 'F']]
1
1
correct
1
correct

```

This should execute when the test is over

Output is not given

## Testing

After some testing, I realise that the condition is never met as the if statement contradicts the while statement that it is contained in. Furthermore, I realise that as an array is initialised from 0 the position of the final question will not match the length of the array, so it needs to be incremented by 1. The following changes are made, and the confirming output is given.

```

i=0

while i!=len(questions):
    if currentQuestion!=questions[i]:
        i=i+1
    elif currentQuestion==questions[i]:
        checkAnswer(answer,answers,i,testName,rowOfEntry,rows)
        if i+1<len(questions):
            newQuestion=questions[i+1]
            questionBox.insert(END,newQuestion)
            break
        break

if i+1==len(questions):
    print("end of test")

```

>>>

```

['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '2', 'F'], ['a', '2', 'F'], ['a', '2', 'F'], ['a', '0', 'F']]
4
4
correct
4
correct
end of test

```

The if statement is clearly met at the end of the test given by the output

Now I can create a function to calculate the grade based on the user's marks and the grade boundaries for the test.

I make the following function

## Testing

```

def updateGrade(testName, rowOfEntry):
    file=testName+" - marks.csv"
    csvFile=open(file, 'r+')
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    print(rows)

```

(function call within updateQuestionBox function)

```
while i!=len(questions):
    if currentQuestion!=questions[i]:
        i=i+1
    elif currentQuestion==questions[i]:
        checkAnswer(answer,answers,i,testName,rowOfEntry,rows)
        if i+1<len(questions):
            newQuestion=questions[i+1]
            questionBox.insert(END,newQuestion)
            break
        break

if i+1==len(questions):
    updateGrade(testName, rowOfEntry)
```

```
>>>
['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '2', 'F'], ['a', '2', 'F'], ['a', '2', 'F'],
 ['a', '2', 'F'], ['a', '0', 'F']]
5
5
correct
5
wrong
wrong
wrong
wrong
[]
```

Wrong answer entered leads to an empty array, which is supposed to store the contents of the csv file

The program runs with no errors, however outputted empty array makes me realise that if a wrong question is entered all the contents of the csv file are deleted.

To fix this I need to make my checkAnswer function rewrite the contents of the csv file back into the csv file before ending the function.

```
def checkAnswer(answer, answers, i, testName, rowOfEntry, rows):
    file=testName+" - marks.csv"
    csvFile=open(file, 'w', newline='')
    csvWriter=csv.writer(csvFile)

    correctAnswers=answers[i]

    print(rowOfEntry)

    j=0

    while j!=len(correctAnswers):
        if answer==correctAnswers[j]:
            rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
            csvWriter.writerows(rows)
            print("correct")
            break
        elif answer!=correctAnswers[j]:
            print("wrong")
            j=j+1

    csvFile.close()

# Checks the inputted answer against the actual answers
def checkAnswer(answer, answers, i, testName, rowOfEntry, rows):
    file=testName+" - marks.csv"
    csvFile=open(file, 'w', newline='')
    csvWriter=csv.writer(csvFile)

    correctAnswers=answers[i]

    print(rowOfEntry)

    j=0

    while j!=len(correctAnswers):
        if answer==correctAnswers[j]:
            rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
            csvWriter.writerows(rows)
            print("correct")
            break
        elif answer!=correctAnswers[j]:
            csvWriter.writerows(rows)
            print("wrong")
            j=j+1

    csvFile.close()
```

Function now rewrites the contents of the csv file even if the answer is incorrect

## Testing

When I test the new code, I see a new problem, the contents of the csv file are rewritten into the csv file multiple times.

```
>>>
['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings have been written
[['Username', 'Marks', 'Grades'], ['a', '0', 'F']]
1
1
wrong
wrong
wrong
1
wrong
wrong
wrong
[['Username', 'Marks', 'Grades'], ['a', '0', 'F'], ['Username', 'Marks', 'Grades'],
 ['a', '0', 'F'], ['Username', 'Marks', 'Grades'], ['a', '0', 'F']]
```

	A	B	C
1	Username	Marks	Grades
2	a	0	F
3	Username	Marks	Grades
4	a	0	F
5	Username	Marks	Grades
6	a	0	F

To fix this I add another condition to the if statement, this makes the program only check if the inputted answer matches the actual answers, until it reaches the final actual correct answer where it will also check if it is wrong, this is because at this point if the inputted answer does not match the final listed correct answer then it must be wrong. I also move the code to increment j outside of this elif statement to ensure that a new answer is always checked.

```
# Checks the inputted answer against the actual answers
def checkAnswer(answer,answers,i,testName,rowOfEntry,rows):
    file=testName+" - marks.csv"
    csvFile=open(file,'w',newline='')
    csvWriter=csv.writer(csvFile)

    correctAnswers=answers[i]

    print(rowOfEntry)

    j=0

    while j!=len(correctAnswers):
        if answer==correctAnswers[j]:
            rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
            csvWriter.writerows(rows)
            print("correct")
            break
        elif answer!=correctAnswers[j] and j+1==len(correctAnswers):
            csvWriter.writerows(rows)
            print("wrong")
        j=j+1

    csvFile.close()

>>>
['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings have been written
[['Username', 'Marks', 'Grades'], ['a', '0', 'F']]
1
1
correct
1
wrong
[['Username', 'Marks', 'Grades'], ['a', '1', 'F']]
```

## Testing

I add some more code to get the marks and get a list of the grade boundaries. The program runs with no errors and all outputs are correct.

```
def updateGrade(testName, rowOfEntry):
    file=testName+" - marks.csv"
    csvFile=open(file, 'r+')
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    csvFile.close()

    print(rows)

    marks=rows[rowOfEntry][1]
    print(marks)

    file=testName+" - grades.csv"
    csvFile=open(file, 'r+')
    csvReader=csv.reader(csvFile)
    grades=list(csvReader)

    print(grades)|

['question1', 'question2']
[['a1', 'a2', 'a3'], ['a4', 'a5', 'a6']]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '1', 'F'], ['a', '0', 'F']]
2
2
correct
2
correct
[['Username', 'Marks', 'Grades'], ['a', '1', 'F'], ['a', '2', 'F']]
2
[['A', 'B', 'C', 'D', 'E', 'F'], ['1', '1', '1', '1', '1', '1']]
```

I need a better sample test to test this function on, so I create a new simple test using the program.

	A	B
1	Question	Answer
2	q1	a1
3	q2	a2
4	q3	a3
5	q4	a4
6	q5	a5
7	q6	a6
8	q7	a7
9	q8	a8
10	q9	a9

	A	B	C	D	E	F
1	A	B	C	D	E	F
2	8	6	4	3	2	1
3						

## Error

Next, I add code to determine the grade based on the user's results and update the csv file accordingly. However, I receive an error.

```
def updateGrade(testName, rowOfEntry):
    file=testName+" - marks.csv"
    csvFile=open(file,'r+')
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    csvFile.close()

    print(rows)

    marks=rows[rowOfEntry][1]
    print(marks)

    file=testName+" - grades.csv"
    csvFile=open(file,'r+')
    csvReader=csv.reader(csvFile)
    grades=list(csvReader)

    print(grades)

    grade=""

    while i!=len(grades[1]):
        if marks>=grades[1][i]:
            grade=grades[1][i]
            break
        else:
            i=i+1|
```

```
>>>
['q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7', 'q8', 'q9']
[[['a1'], ['a2'], ['a3'], ['a4'], ['a5'], ['a6'], ['a7'], ['a8'], ['a9']]]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '0', 'F']]
1
1
correct
9
[['A', 'B', 'C', 'D', 'E', 'F'], ['8', '6', '4', '3', '2', '1']]
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 248, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers,testName,rowOfEntry,rows))
  File "N:\GUI.py", line 208, in updateQuestionBox
    updateGrade(testName,rowOfEntry)
  File "N:\GUI.py", line 174, in updateGrade
    while i!=len(grades[1]):
UnboundLocalError: local variable 'i' referenced before assignment
```

To fix this I define the variable i, I also realise that the grades should be set to U by default, in case a user gets a score under all defined grade boundaries

```
grade=""
i=0

while i!=len(grades[1]):
    if marks>=grades[1][i]:
        grade=grades[1][i]
        break
    else:
        i=i+1
```

```

def setMarkFile(username, testName, questions, answers):
    file=testName+" - marks.csv"
    csvFile=open(file, 'a')
    newLine=username+",0,F\n"
    csvFile.write(newLine)
    csvFile.close()

```

```

def setMarkFile(username, testName, questions, answers):
    file=testName+" - marks.csv"
    csvFile=open(file, 'a')
    newLine=username+",0,U\n"
    csvFile.write(newLine)
    csvFile.close()

```

## Testing

The program now runs with no errors and sets the default grade as U

```

>>>
['q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7', 'q8', 'q9']
[[['a1'], ['a2'], ['a3'], ['a4'], ['a5'], ['a6'], ['a7'], ['a8'], ['a9']]]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '9', 'F'], ['a', '0', 'U']]
2
2
correct
2
correct
2
correct
2
correct
2
correct
2
correct
2
wrong
2
wrong
2
wrong
6
[['A', 'B', 'C', 'D', 'E', 'F'], [8, 6, 4, 3, 2, 1]]

```

Grade is set to U by default

## Error

Next, I add code to save the new contents of the grade csv with the updated grade. However, I get an error.

```
def updateGrade(testName, rowOfEntry):
    file=testName+" - marks.csv"
    csvFile=open(file, 'r+')
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    csvFile.close()

    print(rows)

    marks=rows[rowOfEntry][1]
    print(marks)

    file=testName+" - grades.csv"
    csvFile=open(file, 'r+')
    csvReader=csv.reader(csvFile)
    grades=list(csvReader)

    print(grades)

    grade=""
    i=0

    while i!=len(grades[1]):
        if marks>=grades[1][i]:
            grade=grades[1][i]
            break
        else:
            i=i+1

    print(grade)

    csvFile.close()

    csvFile=open(file, 'w', newline='')
    csvWriter=csv.writer(csvFile)
    csvWriter.writerows(grades)
```

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\GUI.py", line 255, in <lambda>
    submitButton=Button(completeWindow,text="Submit",command=lambda: updateQuestionBox(questionBox,questions,answerBox,answers,testName,rowOfEntry,rows))
  File "N:\GUI.py", line 215, in updateQuestionBox
    updateGrade(testName,rowOfEntry)
  File "N:\GUI.py", line 175, in updateGrade
    while i!=len(grades[1]):
IndexError: list index out of range
```

I fix this by changing the loop to a for loop, this is more suitable because there is a definite maximum number of times that the code must enter the loop. **Otherwise, the code will can loop for ever which causes an index error.**

## Testing

I run the code and receive no errors; however, the grade is still unmodified.

```
for i in range(0,5):
    if marks>=grades[1][i]:
        grade=grades[1][i]
        break
    else:
        i=i+1

print(grade)

csvFile.close()

csvFile=open(file,'w',newline='')
csvWriter=csv.writer(csvFile)
csvWriter.writerows(grades)
```

```

>>>
['q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7', 'q8', 'q9']
[['a1'], ['a2'], ['a3'], ['a4'], ['a5'], ['a6'], ['a7'], ['a8'], ['a9']]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '0', 'U']]
1
1
correct
1
wrong
1
wrong
1
wrong
[['Username', 'Marks', 'Grades'], ['a', '7', 'U']]
7
[['A', 'B', 'C', 'D', 'E', 'F'], [8, 6, 4, 3, 2, 1]]
6

```

A mark of 7 was achieved,  
therefore according to the  
grades file, the user should  
receive a grade B

	A	B	C
1	Username	Marks	Grades
2	a	7	U

To fix this I make several modifications to my code.

(Before)

```
def updateGrade(testName, rowOfEntry):
    file=testName+" - marks.csv"
    csvFile=open(file,'r+')
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    csvFile.close()

    print(rows)

    marks=rows[rowOfEntry][1]
    print(marks)

    file=testName+" - grades.csv"
    csvFile=open(file,'r+')
    csvReader=csv.reader(csvFile)
    grades=list(csvReader)

    print(grades)

    grade=""
    i=0

    for i in range(0,5):
        if marks>=grades[1][i]:
            grade=grades[1][i]
            break
        else:
            i=i+1

    print(grade)

    csvFile.close()

    csvFile=open(file,'w',newline='')
    csvWriter=csv.writer(csvFile)
    csvWriter.writerows(grades)|
```

(After)

```
def updateGrade(testName, rowOfEntry):
    markFile= testName+" - marks.csv"
    csvFile=open(markFile, 'r+')
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    csvFile.close()

    print(rows)

    marks=rows[rowOfEntry][1]
    print(marks)

    gradeFile=testName+" - grades.csv"
    csvFile=open(gradeFile,'r+')
    csvReader=csv.reader(csvFile)
    grades=list(csvReader)

    print(grades)

    grade=""
    i=0

    for i in range(0,5):
        if marks>=grades[1][i]:
            grade=grades[0][i]
            break
        else:
            i=i+1

    print(grade)

    csvFile.close()

    rows[rowOfEntry][2]=grade

    csvFile=open(markFile, 'w', newline=' ')
    csvWriter=csv.writer(csvFile)
    csvWriter.writerows(rows)
```

Changing variable name as two file names are used

Changing variable name as two file names are used

Changing index to get the grade and not the grade boundary

Adding new grade to the marks file and writing the new contents of the marks file

```

>>>
['q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7', 'q8', 'q9']
[['a1'], ['a2'], ['a3'], ['a4'], ['a5'], ['a6'], ['a7'], ['a8'], ['a9']]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '7', 'U'], ['a', '0', 'U']]
2
2
correct
2
wrong
2
|wrong
[[['Username', 'Marks', 'Grades'], ['a', '7', 'U'], ['a', '7', 'U']]]
7
[['A', 'B', 'C', 'D', 'E', 'F'], ['8', '6', '4', '3', '2', '1']]
B

```

	A	B	C
1	Username	Marks	Grades
2	a		7 U
3	a		7 B

New grade is added correctly

## Test results summary GUI

Now, I can make the results GUI, which will display the user's results. **Otherwise, the user cannot know how well they did.**

First, I create the GUI.

```

def ResultsGUI(grade,marks,username,testName,totalMarks):
    results=Tk()
    results.geometry("450x140")
    results.title("Results")
    results.configure(bg="#33BFFF")

    results.resizable(False,False)

    labelResultsMenu=Label(results,text="Your results")
    labelResultsMenu.pack(side=TOP)

    labelTestName=Label(results,bg="#33BFFF",text="TestName: "+testName)
    labelTestName.place(x=5,y=40)

    labelUsername=Label(results,bg="#33BFFF",text="Username: "+username)
    labelUsername.place(x=5,y=60)

    labelMarks=Label(results,bg="#33BFFF",text="Marks: "+str(marks)+"/"+str(totalMarks))
    labelMarks.place(x=340,y=40)

    percentage=100*float(marks)/float(totalMarks)

    labelPercent=Label(results,bg="#33BFFF",text="Percentage: "+str(percentage)+"%")
    labelPercent.place(x=340,y=60)

    labelGrade=Label(results,bg="#33BFFF",text="Grade: "+grade)
    labelGrade.place(x=340,y=80)

    closeButton=Button(results,text="Close",command=lambda: results.destroy())
    closeButton.place(x=340,y=100)

    tryAgainButton=Button(results,text="Try again",command=lambda: readTestFile(testName+".csv","Read",username))
    tryAgainButton.place(x=10,y=100)

```

Then, I add to the updateGrade function to define all the variables that will be needed for the GUI. **Otherwise, an error will appear when the function is called as not all variables in the argument will be defined in the local scope.**

```

rows[rowOfEntry][2]=grade
username=rows[rowOfEntry][0]

csvFile=open(markFile,'w',newline='')
csvWriter=csv.writer(csvFile)
csvWriter.writerows(rows)

csvFile.close()

totalMarks=0

testFile=testName+".csv"
contents=open(testFile)
fileContents=(contents)
next(contents)

print(fileContents)

for row in fileContents:
    totalMarks=totalMarks+1

print(totalMarks)

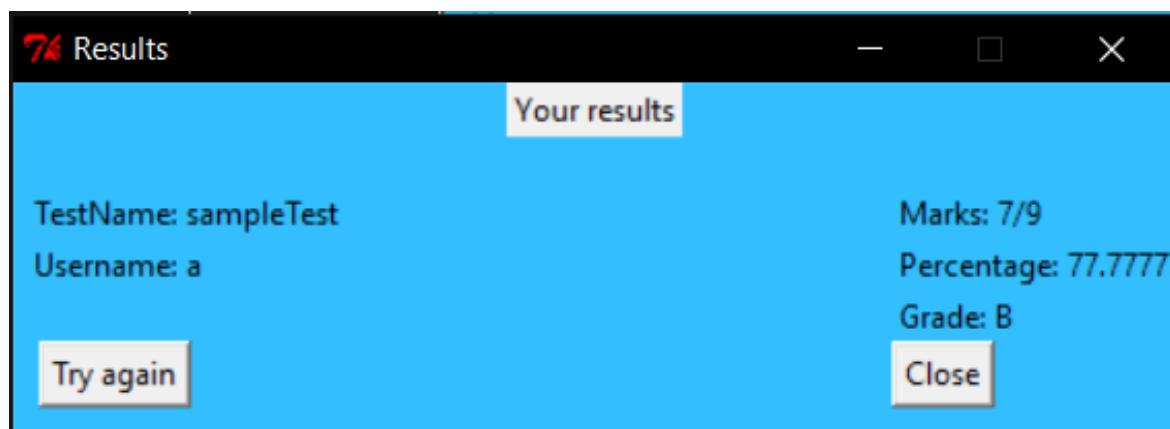
```

Code added to obtain the username and total marks to display these in the results GUI, also the total marks will be needed to calculate the percentage.

```
ResultsGUI(grade,marks,username,testName,totalMarks)
```

## Testing

```
['q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7', 'q8', 'q9']
[['a1'], ['a2'], ['a3'], ['a4'], ['a5'], ['a6'], ['a7'], ['a8'], ['a9']]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '7', 'U'], ['a', '7', 'B'], ['a', '7', 'B'],
 ['a', '7', 'B'], ['a', '0', 'U']]
5
5
correct
5
correct
5
correct
5
correct
5
correct
5
correct
5
wrong
5
wrong
[['Username', 'Marks', 'Grades'], ['a', '7', 'U'], ['a', '7', 'B'], ['a', '7', 'B'],
 ['a', '7', 'B'], ['a', '7', 'U']]
7
[['A', 'B', 'C', 'D', 'E', 'F'], ['8', '6', '4', '3', '2', '1']]
B
<_io.TextIOWrapper name='sampleTest.csv' mode='r' encoding='cp1252'>
9
```



## Error

The program runs correctly, however, when the try again button is pressed an error is given.

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "C:\Users\acer\Desktop\Computing\GUI.py", line 182, in <lambda>
    tryAgainButton=Button(results,text="Try again",command=lambda: readTestFile(
testName+".csv","Read",username))
  File "C:\Users\acer\Desktop\Computing\GUI.py", line 28, in readTestFile
    csvFile=open(testFile,'r+')
IOError: [Errno 2] No such file or directory: 'sampleTest.csv.csv'

```

To fix this, I modify the parameters of the command of the button. I remove the extra ".csv" at the end of the test name because the readTestFile function adds ".csv" to the end of the file name.

(before)

```
tryAgainButton=Button(results,text="Try again",command=lambda: readTestFile(testName+".csv","Read",username))
tryAgainButton.place(x=10,y=100)
```

(after)

```
tryAgainButton=Button(results,text="Try again",command=lambda: readTestFile(testName,"Read",username))
tryAgainButton.place(x=10,y=100)
```

This is the end of my first test completion prototype. Next, I will make the feedback giving feature.

## Feedback giver

### Feedback file creation function

First, I need to create a feedback csv. I will do this by modifying my gradeFileCreator function, which creates both the grades and marks file. I will add code to also create a feedback file. **Otherwise, I will need to create a separate function, which will make the code harder to maintain, increase the likelihood of errors and slow down the development time.**

```

def gradeFileCreator(testName,A,B,C,D,E,F,gradeWindow):
    rootFile=os.path.abspath(os.curdir)
    gradeFileName=testName.strip(".csv")
    dst1File=gradeFileName+" - grades"+".csv"
    dst2File=gradeFileName+" - marks"+".csv"
    rootFile=rootFile+"\empty.csv"

    shutil.copy2(rootFile,dst1File)
    shutil.copy2(rootFile,dst2File)

    openGradeFile(dst1File,A,B,C,D,E,F,gradeWindow)
    openMarkFile(dst2File)

```

```

def gradeFileCreator(testName,A,B,C,D,E,F,gradeWindow):
    rootFile=os.path.abspath(os.curdir)
    gradeFileName=testName.strip(".csv")
    dst1File=gradeFileName+" - grades.csv"
    dst2File=gradeFileName+" - marks.csv"
    dst3File=gradeFileName+" - feedback.csv"
    rootFile=rootFile+"\empty.csv"

    shutil.copy2(rootFile,dst1File)
    shutil.copy2(rootFile,dst2File)
    shutil.copy2(rootFile,dst3File)

    openGradeFile(dst1File,A,B,C,D,E,F,gradeWindow)
    openMarkFile(dst2File)
    openFeedbackFile(dst3File)

```

## Error

## Testing

I run the program and create a new test to make sure this new code works. The file is created successfully, but an error appears

```

>>>
C:\Users\acer\Desktop\Computing\test1234.csv
headings have been written
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "C:\Users\acer\Desktop\Computing\GUI.py", line 482, in <lambda>
    submitButton=Button(gradeWindow,text="Submit",command=lambda: gradeFileCreator(file,entryAgrade,entryBgrade,entryCgrade,entryDgrade,entryEgrade,entryFgrade,gradeWindow))
  File "C:\Users\acer\Desktop\Computing\GUI.py", line 411, in gradeFileCreator
    openFeedbackFile(dst3File)
NameError: global name 'openFeedbackFile' is not defined

```

- ❑ test1234 - feedback.csv
- ❑ test1234 - grades.csv
- ❑ test1234 - marks.csv
- ❑ test1234.csv

Next, I should define the openFeedbackFile function. The purpose of this function is to add the headings to the csv file. However, I realise that this is like my openMarkFile function, therefore I will turn this function into a new function that is usable by both files.

First, I remove some obsolete code that I find. **Without this, the function will take longer to execute making the program slower for a user to use.**

```
# Opens the mark file and formats the headings
def openMarkFile(file):
    csvFile=open(file,'r+')
    fileRead=csvFile.read()
    if fileRead.strip()=='':
        fileRead=csvFile.read()
        csvFile=open(file,'w+')
        csvFile.write('Username,Marks,Grades\n')

    for item in fileRead:
        csvFile.write('{Username},{Marks},{Grades}'.format(**item))
        csvFile.write('\n')
    csvFile.close()

    print("headings have been written")
else:
    print("headings are already written")

def openFeedbackFile(file):
    csvFile=open(file,'r+')
    fileRead=csvFile.read()
    if fileRead.strip()=='':
        fileRead=csvFile.read()
        csvFile=open(file,'w+')
        csvFile.write('Username,Marks,Grades\n')

        print("headings have been written")
    else:
        print("headings are already written")
```

This code is not necessary as the headers are written by the line above

Obselete code has been removed

Next, I modify the code to work with both the feedback and mark files.

```

def openMarkOrFeedbackFile(fileName, fileType):
    csvFile=open(fileName, 'r+')
    fileRead=csvFile.read()
    if fileRead.strip()=='' and fileType=="Mark":
        fileRead=csvFile.read()
        csvFile=open(fileName, 'w+')
        csvFile.write('Username,Marks,Grades\n')
        print("headings have been written")
    elif fileRead.strip()=='' and fileType=="Feedback":
        fileRead=csvFile.read()
        csvFile=open(fileName, 'w+')
        csvFile.write('Username,Feedback\n')
        print("headings have been written")
    else:
        print("headings are already written")

```

Function name and variable names changed to match function's new purpose.

New code added to add different headings for feedback file

Then I change the function calls to match the new names and parameters.

```

def gradeFileCreator(testName,A,B,C,D,E,F,gradeWindow):
    rootFile=os.path.abspath(os.curdir)
    gradeFileName=testName.strip(".csv")|
    dst1File=gradeFileName+" - grades.csv"
    dst2File=gradeFileName+" - marks.csv"
    dst3File=gradeFileName+" - feedback.csv"
    rootFile=rootFile+"\empty.csv"

    shutil.copy2(rootFile,dst1File)
    shutil.copy2(rootFile,dst2File)
    shutil.copy2(rootFile,dst3File)

    openGradeFile(dst1File,A,B,C,D,E,F,gradeWindow)
    openMarkFile(dst2File)
    openFeedbackFile(dst3File)

```

```
def gradeFileCreator(testName,A,B,C,D,E,F,gradeWindow):
    rootFile=os.path.abspath(os.curdir)
    gradeFileName=testName.strip(".csv")
    dst1File=gradeFileName+" - grades.csv"
    dst2File=gradeFileName+" - marks.csv"
    dst3File=gradeFileName+" - feedback.csv"
    rootFile=rootFile+"\empty.csv"

    shutil.copy2(rootFile,dst1File)
    shutil.copy2(rootFile,dst2File)
    shutil.copy2(rootFile,dst3File)

openGradeFile(dst1File,A,B,C,D,E,F,gradeWindow)
openMarkOrFeedbackFile(dst2File,"Mark")
openMarkOrFeedbackFile(dst3File,"Feedback")|
```

```
def readAllQuestions(testName,testFile,username):
    questions=[]
    answers=[]
    temp=[]

    csvFile=open(testFile)
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    i=1
    while i!=len(rows):
        question=rows[i][0]
        questions.append(question)
        i=i+1

    print(questions)

    i=1
    j=1
    while i!=len(rows):
        while j!=len(rows[i]):
            answer=rows[i][j]
            temp.append(answer)
            j=j+1
        answers.append(temp)
        temp=[]
        j=1
        i=i+1

    print(answers)

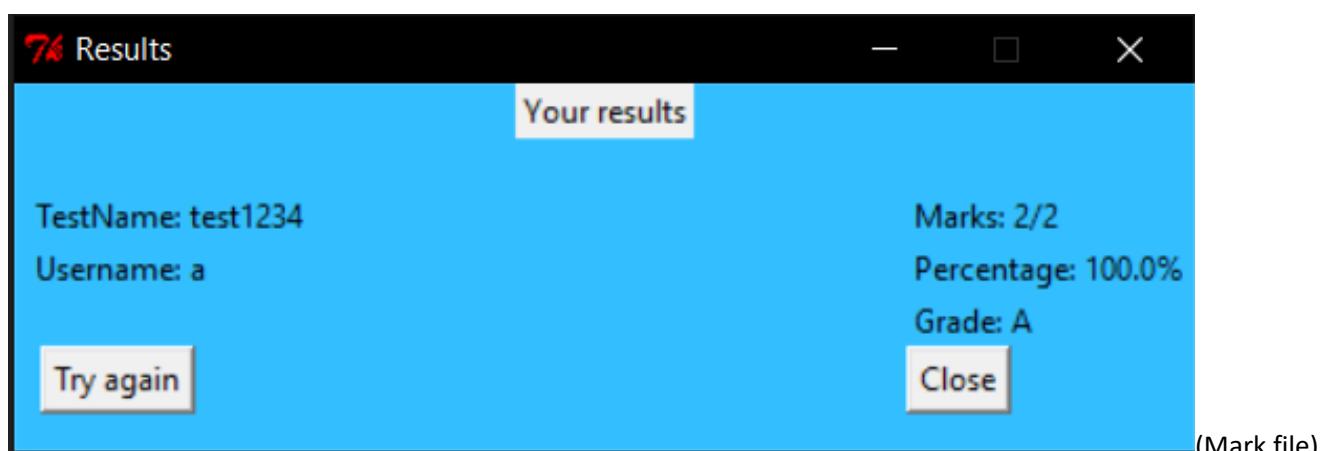
openMarkFile(testName+' - marks.csv')
    setMarkFile(username,testName,questions,answers)
```

Old function call

## Testing

I run the program, both creating a test and completing the test to ensure that the files are still usable by the necessary functions throughout the program. The program runs with no errors.

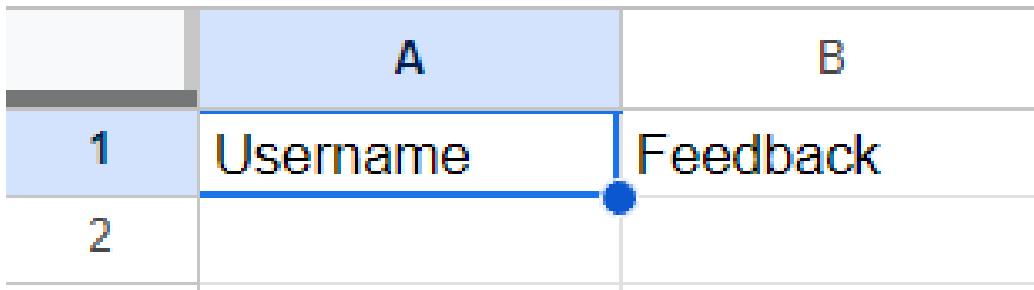
```
>>>
C:\Users\acer\Desktop\Computing\test1234.csv
headings have been written
headings have been written
['a', 'b']
[['a'], ['b']]
headings are already written
[['Username', 'Marks', 'Grades'], ['a', '0', 'U']]
1
1
correct
1
correct
|[['Username', 'Marks', 'Grades'], ['a', '2', 'U']]
2
[['A', 'B', 'C', 'D', 'E', 'F'], ['2', '1', '1', '1', '1', '1']]
A
<_io.TextIOWrapper name='test1234.csv' mode='r' encoding='cp1252'>
2
```



	A	B	C
1	Username	Marks	Grades
2	a	2	A

Mark file headings added correctly and all functions to calculate and append to the file accordingly have been done successfully

(Feedback file)



Feedback file has been created successfully  
and its specific headings have been added.

### Type of feedback selector GUI

Next, I can create the first GUI to select what type of feedback should be given. **Otherwise, the user cannot select who they want to give feedback to.**

```
#  
#   FEEDBACK GIVER  
#  
  
def feedbackSelectorGUI():  
    feedbackSelector=Tk()  
    feedbackSelector.geometry("450x160")  
    feedbackSelector.title("Feedback sender")  
    feedbackSelector.configure(bg="#33BFFF")  
  
    feedbackSelector.resizable(False,False)  
  
    labelFeedbackMenu=Label(feedbackSelector,text="Feedback sender")|  
    labelFeedbackMenu.pack(side=TOP)  
  
    specificButton=Button(feedbackSelector,width=15,height=3,wraplength=80,text="Give feedback to a specific student")  
    specificButton.place(x=10,y=40)  
  
    wholeButton=Button(feedbackSelector,width=15,height=3,wraplength=80,text="Give feedback on a whole test")  
    wholeButton.place(x=320,y=40)  
  
    closeButton=Button(feedbackSelector,text="Close",command=lambda: feedbackSelector.destroy())  
    closeButton.place(x=360,y=130)
```

```

def secondaryMenu(username,adminStatus):
    sec=Tk()
    sec.geometry("450x50")
    sec.title("Features Menu")
    sec.configure(bg="#33BFFF")

    sec.resizable(False,False)

    labelSecMenu=Label(sec,text="Welcome to the program")
    labelSecMenu.pack(side=TOP)

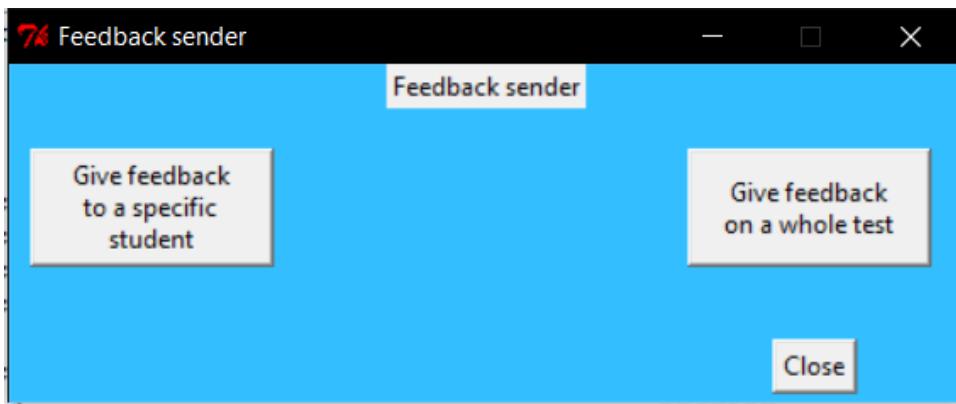
    buttonComplete=Button(sec,text="Complete a test",command=lambda: nameTestFile("Complete a
buttonComplete.place(x=10, y=20)

    buttonRFeedback=Button(sec,text="Recieve feedback")
    buttonRFeedback.place(x=330,y=20)

    if adminStatus==True:
        sec.geometry("450x200")
        buttonGFeedback=Button(sec,text="Give feedback",command=lambda: feedbackSelectorGUI())
        buttonGFeedback.place(x=330,y=150)

```

Makes button execute the function.  
**Otherwise, the user would not be able to open the GUI if they want to give feedback**



### Get all usernames and test names function

Next, I will make a function to get all the registered usernames and test names. **Otherwise, the user would find it difficult to find the name of the user or test that they would like to give feedback on.**

```

# Gets usernames and test names for button functions in the next GUI
def getUsernameNames(function):
    # Gets usernames using csv file
    if function=="Specific":
        csvFile=open('login.csv','r+')
        csvReader=csv.reader(csvFile)
        rows=list(csvReader)

        usernamesList=[]

        print(rows)

        for i in range(1,len(rows)):
            username=rows[i][0]
            usernamesList.append(username)

        csvFile.close()

        nameFeedbackReciever("Specific",usernamesList)

    # Gets usernames using os module, somewhat copied from
    if function=="Test":
        rootFile=os.path.abspath(os.curdir)
        listOfFile=os.listdir(rootFile)
        print(listOfFile)

        testNamesList=[]

        # Only saves files if they end in "grades.csv" as :
        for i in range(0,len(listOfFile)):
            file=listOfFile[i]
            if file[-10:]=="grades.csv":
                testNamesList.append(file[0:-13])

        print(testNamesList)

        nameFeedbackReciever("Test",testNamesList)

```

I use if statements and a parameter to only display the GUI and information required based on what option the user selected previously. **Otherwise, the user would not be able to choose who they want to send feedback to**

A second GUI will be opened (in the yet to be defined nameFeedbackReciever function) and formatted based on the option selected by the user on the previous GUI. **Otherwise, information that is not required by the user will be shown, this makes the program less efficient.**

Then, I add an extra print statement that I forgot to add. **Otherwise, I will not be able to test the program at this stage.**

```

def getUsernameAndTestNames(function):
    # Gets usernames using csv file
    if function=="Specific":
        csvFile=open('login.csv','r+')
        csvReader=csv.reader(csvFile)
        rows=list(csvReader)

        usernamesList=[]

        print(rows)

        for i in range(1,len(rows)):
            username=rows[i][0]
            usernamesList.append(username)

        print(usernamesList) | print(usernamesList)

        csvFile.close()

    nameFeedbackReciever("Specific",usernamesList)

```

Print statement added so I can test that the array has the correct information required

Now, I test the function making sure I go through both if statements. I also temporarily remove the nameFeedbackReciever function. Otherwise, **an error will be receive as the function has not been yet defined.**

When the “Give feedback to a specific student” button is pressed the program outputs the following, which matches the usernames in the login function.

```

>>>
[['Username', 'Password', 'Admin'], ['testusername', 'testpassword', 'testadmin'],
 ['noadmin1', 'noadmin2', ''], ['a', 'a', 'testadmin'], ['input1', 'input2',
 'input3'], ['a', '', ''], ['', 'a', '']]
['testusername', 'noadmin1', 'a', 'input1', 'a', '']

```

usernameList's contents

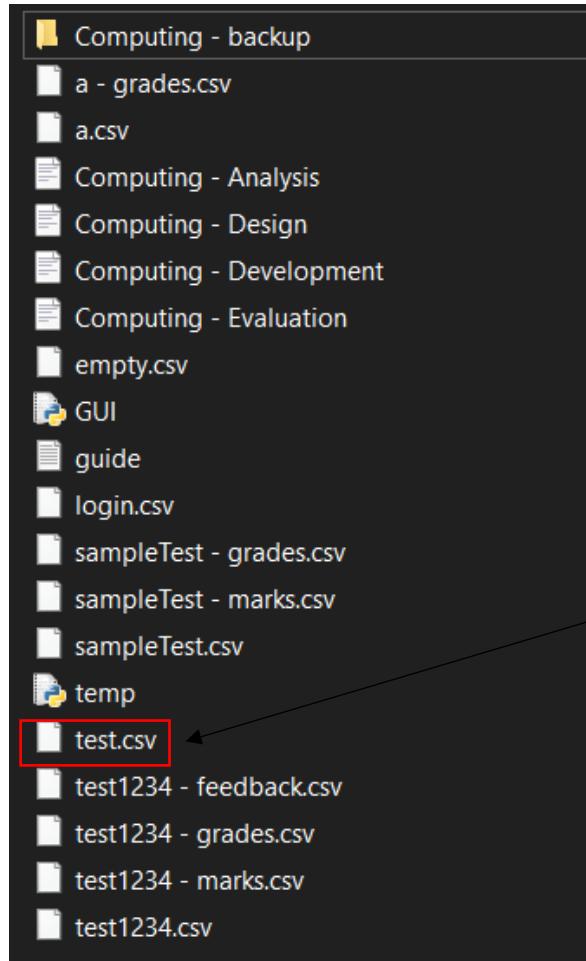
(login.csv file contents)

	A	B	C
1	Username	Password	Admin
2	testusername	testpassword	testadmin
3	noadmin1	noadmin2	
4	a	a	testadmin
5	input1	input2	input3
6	a		
7		a	
8			

When the “Give feedback on a whole test” button is pressed the program outputs the following, which matches the test names in my directory

```
['a - grades.csv', 'a.csv', 'Computing - Analysis.docx', 'Computing - backup', 'Computing - Design.docx', 'Computing - Development.docx', 'Computing - Evaluation.docx', 'empty.csv', 'GUI.py', 'guide.txt', 'login.csv', 'sampleTest - grades.csv', 'sampleTest - marks.csv', 'sampleTest.csv', 'temp.py', 'test.csv', 'test1234 - feedback.csv', 'test1234 - grades.csv', 'test1234 - marks.csv', 'test1234.csv']  
['a', 'sampleTest', 'test1234']
```

testNameList's contents



test.csv does not show up in the array as it is an incomplete test as shown by the lack of a matching grades file.

## Name feedback receiver GUI

Next, I will create a GUI for entering either the test name or the user’s name. I will create a single function/GUI for this, which will change depending on which option is picked. Like my nameTestFile function, which works for both entering the name of a test for completion or creation. **Otherwise, I will need to create two separate functions, which will make the code longer making it harder to maintain.**

I will use the function below as a guide to create this GUI

```

def nameTestFile(title,function,username):
    fileCreator=Tk()
    fileCreator.geometry("450x42")
    fileCreator.title(title)
    fileCreator.configure(bg="#33BFFF")

    fileCreator.resizable(False,False)

    labelCreator=Label(fileCreator,text="Enter the name of a test:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
    labelCreator.place(x=22,y=5)

    nameOfTestBox=Entry(fileCreator,width=52)
    nameOfTestBox.place(x=22,y=20)

    if function=="Create":
        submitButton=Button(fileCreator,text="Submit",command=lambda:testFileCreator(nameOfTestBox,fileCreator))
        submitButton.place(x=375,y=13)
    elif function=="Complete":
        submitButton=Button(fileCreator,text="Submit",command=lambda:checkFileName(nameOfTestBox,username))
        submitButton.place(x=375,y=13)

```

Created GUI. I have used similar if statements to define the label of the input box, the title and the print button. **Otherwise, the names of these may not match the GUI's purpose depending on the type of feedback the user wants to give.**

```

def nameFeedbackReciever(function,listToPrint):
    nameFeedbackReciever=Tk()
    nameFeedbackReciever.geometry("350x90")
    nameFeedbackReciever.configure(bg="#33BFFF")

    nameFeedbackReciever.resizable(False,False)

    nameOfFeedbackReciever=Entry(nameFeedbackReciever,width=52)
    nameOfFeedbackReciever.place(x=22,y=22)

    if function=="Specific":
        nameFeedbackReciever.title("Username to give feedback to")

        labelFunction=Label(nameFeedbackReciever,text="Enter the name of a user:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
        labelFunction.place(x=22,y=5)

        submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda: specificFeedback(nameOfFeedbackReciever))
        submitButton.place(x=270,y=52)

        printButton=Button(nameFeedbackReciever,text="Print all usernames",command=lambda: printList("Specific",listToPrint))
        printButton.place(x=30,y=52)

    elif function=="Test":
        nameFeedbackReciever.title("Test name to give feedback on")

        labelFunction=Label(nameFeedbackReciever,text="Enter the name of a test:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
        labelFunction.place(x=22,y=5)

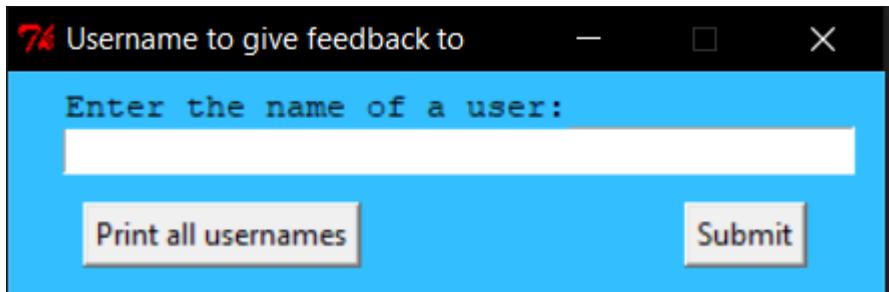
        submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda: wholeFeedback(nameOfFeedbackReciever))
        submitButton.place(x=270,y=52)

        printButton=Button(nameFeedbackReciever,text="Print all test names",command=lambda: printList("Test",listToPrint))
        printButton.place(x=30,y=52)

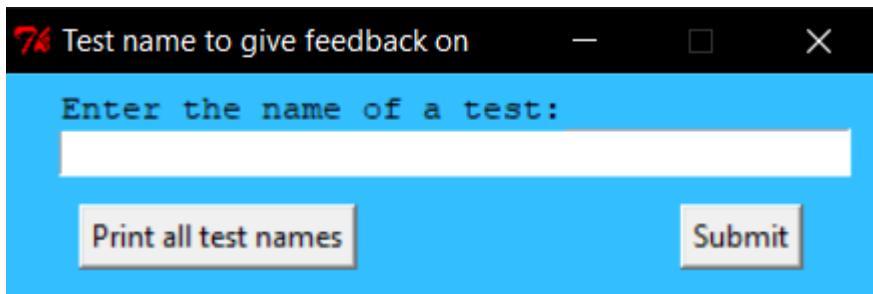
```

I temporarily remove all the commands of the buttons to test the GUI. **Otherwise, I would receive an error as the function have yet to be defined.**

When the “Give feedback to a specific student” button is pressed the program displays the following GUI.



When the “Give feedback on a whole test” button is pressed the program displays the following GUI.



### Display all usernames/test names GUI

Both GUIs have been displayed correctly, next I will define and test the printList function.

I create the following function. Again, I use if statements and the listToPrint parameter to make this function work when displaying either a list of usernames or a list of test names. **Otherwise, I will need to create two separate functions, which will make the code longer making it harder to maintain.**

```
def printList(function,listToPrint):
    displayList=Tk()
    displayList.geometry("450x200")
    displayList.title("Username list")
    displayList.configure(bg="#33BFFF")

    displayList.resizable(False,False)

    scrollbar=Scrollbar(displayList)
    scrollbar.pack(side=RIGHT,fill=Y)

    listToDisplay=Text(displayList,bg="#33BFFF", width=450,wrap=WORD)
    listToDisplay.pack()

    listToDisplay.insert(END,listToPrint)

    listToDisplay.config(yscrollcommand=scrollbar.set,state=DISABLED)
    scrollbar.config(command=listToDisplay.yview)

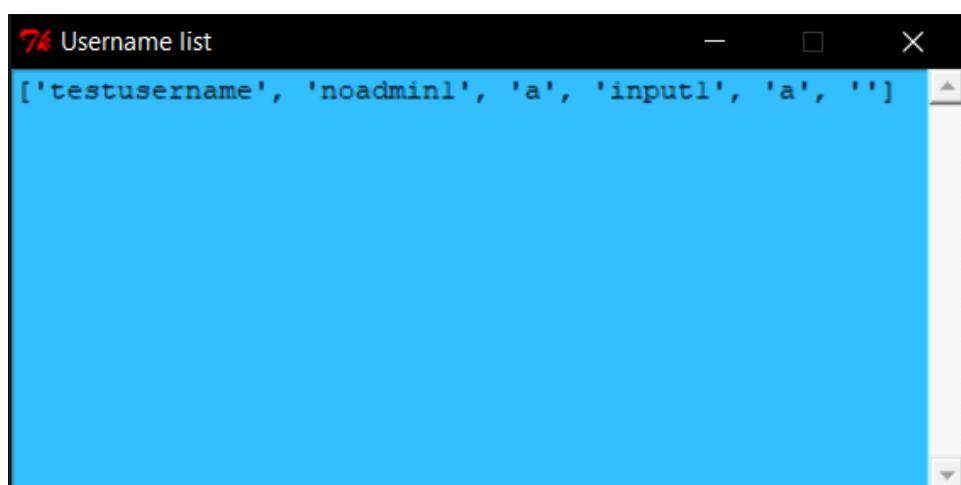
    if function=="Specific":
        displayList.title("Username list")

    elif function=="Test":
        displayList.title("Test names list")
```

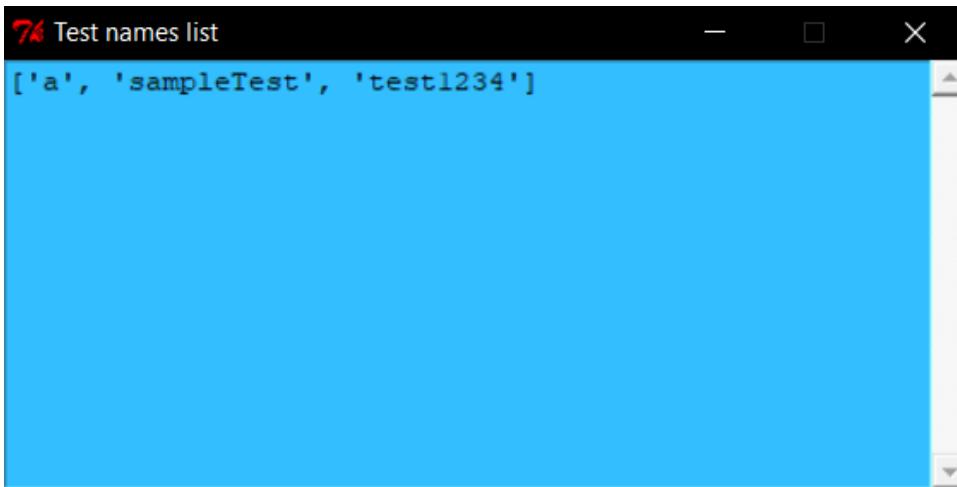
## Testing

Next, I test the GUI with both options.

When the “Give feedback to a specific student” button is pressed the program displays the following GUI.



When the “Give feedback on a whole test” button is pressed the program displays the following GUI.



These still match the print statements getUsernamesAndTestNames function, therefore the GUI's are displaying everything correctly.

### Check inputted feedback receiver

Next, I will work on the submit button. First, I will create a function to check the input against the given list and display an appropriate message box if the user's input is accepted or declined. **Otherwise, the user will not be able to tell if they have entered an invalid username or test name.**

```
def checkFeedbackReciever(function,nameOfFeedbackReciever,listToPrint):
    for i in range(0,len(listToPrint)):
        if function=="Specific":
            if nameOfFeedbackReciever==listToPrint[i]:
                specificFeedback(nameOfFeedbackReciever)
                # define function
        elif nameOfFeedbackReciever!=listToPrint[i] and i==len(listToPrint):
            messagebox.showerror("User has not been found","The entered user has not been found, please try again")

        elif function=="Test":
            if nameOfFeedbackReciever==listToPrint[i]:
                wholeFeedback(nameOfFeedbackReciever)
                # define function
        elif nameOfFeedbackReciever!=listToPrint[i] and i==len(listToPrint):
            messagebox.showerror("Test has not been found","The entered test has not been found, please try again")
```

Now, I need to change the submit buttons commands execute this function instead.

(Before)

```

if function=="Specific":
    nameFeedbackReciever.title("Username to give feedback to")

labelFunction=Label(nameFeedbackReciever,text="Enter the name of a user:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
labelFunction.place(x=22,y=5)

submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda: specificFeedback(nameOfFeedbackReciever))
submitButton.place(x=270,y=52)

printButton=Button(nameFeedbackReciever,text="Print all usernames",command=lambda: printList("Specific",listToPrint))
printButton.place(x=30,y=52)

elif function=="Test":
    nameFeedbackReciever.title("Test name to give feedback on")

labelFunction=Label(nameFeedbackReciever,text="Enter the name of a test:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
labelFunction.place(x=22,y=5)

submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda: wholeFeedback(nameOfFeedbackReciever))
submitButton.place(x=270,y=52)

printButton=Button(nameFeedbackReciever,text="Print all test names",command=lambda: printList("Test",listToPrint))
printButton.place(x=30,y=52)

```

(After, updated commands)

```

if function=="Specific":
    nameFeedbackReciever.title("Username to give feedback to")

labelFunction=Label(nameFeedbackReciever,text="Enter the name of a user:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
labelFunction.place(x=22,y=5)

submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda: checkFeedbackReciever("Specific",nameOfFeedbackReciever,listToPrint))
submitButton.place(x=270,y=52)

printButton=Button(nameFeedbackReciever,text="Print all usernames",command=lambda: printList("Specific",listToPrint))
printButton.place(x=30,y=52)

elif function=="Test":
    nameFeedbackReciever.title("Test name to give feedback on")

labelFunction=Label(nameFeedbackReciever,text="Enter the name of a test:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
labelFunction.place(x=22,y=5)

submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda: checkFeedbackReciever("Test",nameOfFeedbackReciever,listToPrint))
submitButton.place(x=270,y=52)

printButton=Button(nameFeedbackReciever,text="Print all test names",command=lambda: printList("Test",listToPrint))
printButton.place(x=30,y=52)

```

## Error

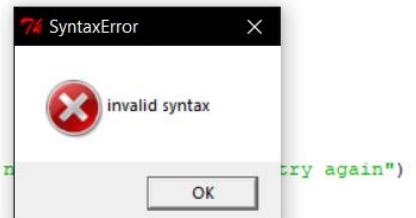
When attempting to run the program, I receive an error message.

```

def checkFeedbackReciever(function,nameOfFeedbackReciever,listToPrint):
    for i in range(0,len(listToPrint)):
        if function=="Specific":
            if nameOfFeedbackReciever==listToPrint[i]:
                specificFeedback(nameOfFeedbackReciever)
                # define function
        elif nameOfFeedbackReciever!=listToPrint[i] and i==len(listToPrint):
            messagebox.showerror("User has not been found","The entered user has not been found, please try again")

        elif function=="Test":
            if nameOfFeedbackReciever==listToPrint[i]:
                wholeFeedback(nameOfFeedbackReciever)
                # define function
            elif nameOfFeedbackReciever!=listToPrint[i] and i==len(listToPrint):
                messagebox.showerror("Test has not been found","The entered test has not been found, please try again")

```



I fix this by modifying the if statement. **Otherwise, the syntax of the statement would be incorrect, leading to an error when the program is ran.**

```
def checkFeedbackReciever(function,nameOfFeedbackReciever,listToPrint):
    for i in range(0,len(listToPrint)):
        if function=="Specific":
            if nameOfFeedbackReciever==listToPrint[i]:
                specificFeedback(nameOfFeedbackReciever)
                # define function
        elif nameOfFeedbackReciever!=listToPrint[i] and i==len(listToPrint):
            messagebox.showerror("User has not been found","The entered user has not been found, please try again")

        elif function=="Test":
            if nameOfFeedbackReciever==listToPrint[i]:
                wholeFeedback(nameOfFeedbackReciever)
                # define function
        elif nameOfFeedbackReciever!=listToPrint[i] and i==len(listToPrint):
            messagebox.showerror("Test has not been found","The entered test has not been found, please try again")
```

## Testing

However, when I run the program no output is given when the submit button is pressed. To fix this I use the .get() function on the nameOfFeedbackReciever variable. **Otherwise, the input stored at nameOfFeedbackReciever cannot be accessed by the program.** I assign the input to a different variable name and change the if statements and arguments accordingly.

(before)

```
def checkFeedbackReciever(function,nameOfFeedbackReciever,listToPrint):
    for i in range(0,len(listToPrint)):
        if function=="Specific":
            if nameOfFeedbackReciever==listToPrint[i]:
                specificFeedback(nameOfFeedbackReciever)
                # define function
        elif nameOfFeedbackReciever!=listToPrint[i] and i==len(listToPrint):
            messagebox.showerror("User has not been found","The entered user has not been found, please try again")

        elif function=="Test":
            if nameOfFeedbackReciever!=listToPrint[i]:
                wholeFeedback(nameOfFeedbackReciever)
                # define function
        elif nameOfFeedbackReciever!=listToPrint[i] and i==len(listToPrint):
            messagebox.showerror("Test has not been found","The entered test has not been found, please try again")
```

(after)

```

def checkFeedbackReciever(function,nameOfFeedbackReciever,listToPrint):
    name=nameOfFeedbackReciever.get()
    for i in range(0,len(listToPrint)):
        if function=="Specific":
            if name==listToPrint[i]:
                specificFeedback(name)
                # define function
        elif name!=listToPrint[i] and i==len(listToPrint):
            messagebox.showerror("User has not been found","The entered user has not been found")
        elif function=="Test":
            if name==listToPrint[i]:
                wholeFeedback(name)
                # define function
        elif name!=listToPrint[i] and i==len(listToPrint):
            messagebox.showerror("Test has not been found","The entered test has not been found")

```

Input stored in the nameOfFeedbackReciever entry widget is assigned to name variable.  
Otherwise, the input cannot be accessed by the program.

Using some print statements, I realise I forgot to add break statements once the name has been found, I need to also reduce the index on the if statement for the error message to appear and I need an if statement to check if the input is blank. **Without the break statement the loop will continue unnecessarily, making the program slower, without reducing the error message the user will never be indicated if their input is invalid, and without checking if the input is blank the loop will be entered unnecessarily, also making the program slower**

```

def specificFeedback(nameOfUser):
    print("found")

def wholeFeedback(nameOfTest):
    print("found")

def checkFeedbackReciever(function,nameOfFeedbackReciever,listToPrint):
    name=nameOfFeedbackReciever.get()
    print(name)

    if name=="":
        messagebox.showerror("Invalid input","Please enter your input into the input box")
    else:
        for i in range(0,len(listToPrint)):
            print(i)
            print(listToPrint[i])
            if function=="Specific":
                if name==listToPrint[i]:
                    specificFeedback(name)
                    break
            elif name!=listToPrint[i] and i==len(listToPrint)-1:
                messagebox.showerror("User has not been found","The entered user has not been found, please try again")
            elif function=="Test":
                if name==listToPrint[i]:
                    wholeFeedback(name)
                    break
            elif name!=listToPrint[i] and i==len(listToPrint)-1:
                messagebox.showerror("Test has not been found","The entered test has not been found, please try again")

```

Ensures loop is not entered if the input is empty

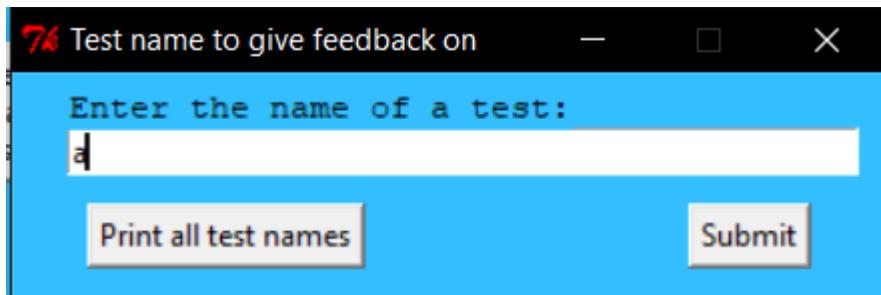
Ensures the loop ends once the name has been confirmed

Ensures error message can appear

## Testing

Then, I test the program to make sure that the function works correctly

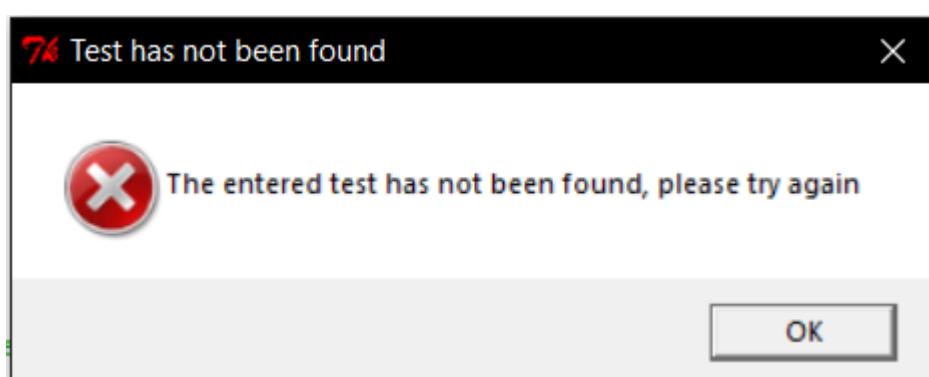
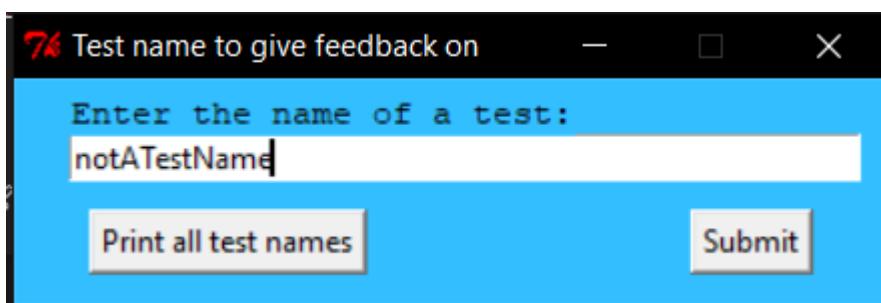
When I enter an existing name, the loop breaks, and the next function is executed correctly.



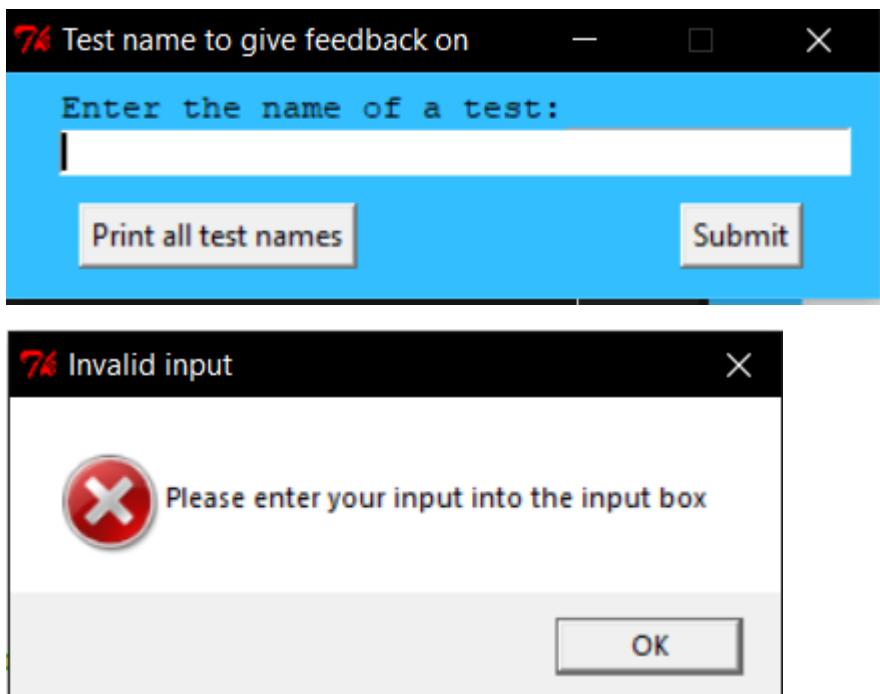
```
[ 'a - grades.csv', 'a.csv', 'Computing - Analysis.docx', 'Computing - backup', 'Computing - Design.docx', 'Computing - Development.docx', 'Computing - Evaluation.docx', 'empty.csv', 'GUI.py', 'guide.txt', 'login.csv', 'sampleTest - grades.csv', 'sampleTest - marks.csv', 'sampleTest.csv', 'temp.py', 'test.csv', 'test1234 - feedback.csv', 'test1234 - grades.csv', 'test1234 - marks.csv', 'test1234.csv' ]
['a', 'sampleTest', 'test1234']
a
0
a
found
```

Output of the next function

When I enter a name that does not exist, the correct error message is given. **Without this, the user would not know why their input has not been accepted.**



When I press the submit button, with an empty input box, the correct error message is given. **Without this, the user would not know why their input has not been accepted.**



Next, I need to work on a GUI that allows the user to enter their feedback. I also realise, this GUI can be executed using 1 function instead of the current 2 functions. So, remove the current 2 functions and replace them with 1 function.

These functions will be replaced.

```
def specificFeedback(nameOfUser):
    print("found")

def wholeFeedback(nameOfTest):
    print("found")
```

The new function

```
def giveFeedbackGUI(function, name):
    print("found")
```

As only 1 function is now called in the checkFeedbackReceiver function, I can shorten the code by removing multiple if statements. **Without this, the program will run slower as more comparisons are required, making it less efficient.**

(before)

```
for i in range(0, len(listToPrint)):
    print(i)
    print(listToPrint[i])
    if function=="Specific":
        if name==listToPrint[i]:
            specificFeedback(name)
            break

    elif name!=listToPrint[i] and function=="Specific":
        messagebox.showerror("User has not been found", "The entered user has not been found, please try again")

    elif function=="Test":
        if name==listToPrint[i]:
            wholeFeedback(name)
            break
```

(after)

```
def checkFeedbackReciever(function, nameOfFeedbackReciever, listToPrint):
    name=nameOfFeedbackReciever.get()

    if name=="":
        messagebox.showerror("Invalid input", "Please enter your input into the input box")
    else:
        for i in range(0, len(listToPrint)):
            if name==listToPrint[i]:
                giveFeedbackGUI(function, name)
                break

            elif name!=listToPrint[i] and function=="Specific" and i==len(listToPrint)-1:
                messagebox.showerror("User has not been found", "The entered user has not been found, please try again")

            elif name!=listToPrint[i] and function=="Test" and i==len(listToPrint)-1:
                messagebox.showerror("Test has not been found", "The entered test has not been found, please try again")
```

## Give feedback GUI

Next, I will work on the giveFeedbackGUI function. **Otherwise, the user does not have a GUI to enter the feedback**

```

def giveFeedbackGUI(function,name):
    giveFeedbackGUI=Tk()
    giveFeedbackGUI.geometry("450x250")
    giveFeedbackGUI.title("Enter feedback to give")
    giveFeedbackGUI.configure(bg="#33BFFF")

    giveFeedbackGUI.resizable(False,False)

    labelFeedbackSender=Label(giveFeedbackGUI,text="Feedback sender")
    labelFeedbackSender.pack(side=TOP)

    labelFunction=Label(giveFeedbackGUI,text="Enter feedback to give:",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0)
    labelFunction.place(x=20,y=22)

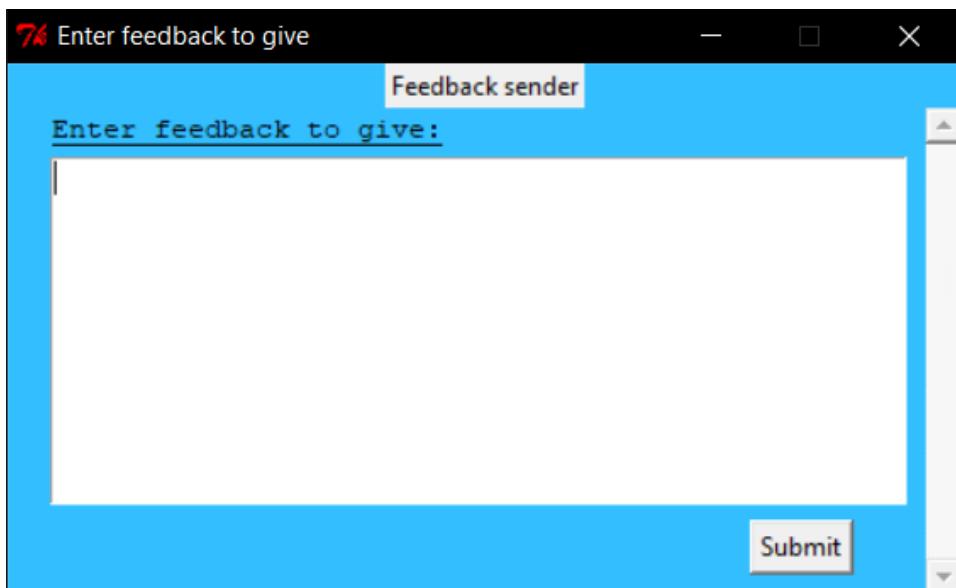
    feedback=Text(giveFeedbackGUI,width=50,height=10,wrap=WORD)
    feedback.place(x=20,y=44)

    submitButton=Button(giveFeedbackGUI,text="Submit",command=lambda: saveFeedback(function,name,feedback))
    submitButton.place(x=350,y=215)

    scrollbar=Scrollbar(giveFeedbackGUI)
    scrollbar.pack(side=RIGHT,fill=Y)

    feedback.config(yscrollcommand=scrollbar.set)
    scrollbar.config(command=feedback.yview)

```



## Name feedback receiver GUI (corrections)

Next, I will work on the saveFeedback function. **Otherwise, the feedback entered into the GUI will not be saved therefore it will not be visible by the receiver.**

However, I realise that when saving specific user feedback, the test name is also required. To fix this I will need to modify several functions.

First, I will modify my getUsersnamesAndTestNames function to also pass the testNamesList variable into the nameFeedbackReciever function

(before)

```
def getUsernameAndTestNames(function):
    if function=="Specific":
        csvFile=open('login.csv','r+')
        csvReader=csv.reader(csvFile)
        rows=list(csvReader)

        usernamesList=[]

        print(rows)

        for i in range(1,len(rows)):
            username=rows[i][0]
            usernamesList.append(username)

        print(usernamesList)

        csvFile.close()

        nameFeedbackReciever("Specific",usernamesList)

    if function=="Test":
        rootFile=os.path.abspath(os.curdir)
        listOfFiles=os.listdir(rootFile)
        print(listOfFiles)

        testNamesList=[]

        for i in range(0,len(listOfFiles)):
            file=listOfFiles[i]
            if file[-10:]=="grades.csv":
                testNamesList.append(file[0:-13])

        print(testNamesList)

        nameFeedbackReciever("Test",testNamesList)
```

(after)

```
def getUsernameAndTestNames(function):
    rootFile=os.path.abspath(os.curdir)
    listOfFiles=os.listdir(rootFile)
    print(listOfFiles)

    testNamesList=[]

    for i in range(0,len(listOfFiles)):
        file=listOfFiles[i]
        if file[-10:]=="grades.csv":
            testNamesList.append(file[0:-13])

    print(testNamesList)
```

Moved outside is statement so that the testNamesList variable is always defined, as it is used for both specific user and whole class feedback

```
if function=="Test":
    nameFeedbackReciever("Test",testNamesList)

if function=="Specific":
    csvFile=open('login.csv','r+')
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    usernamesList=[]

    print(rows)

    for i in range(1,len(rows)):
        username=rows[i][0]
        usernamesList.append(username)

    print(usernamesList)

    csvFile.close()

    nameFeedbackReciever("Specific",usernamesList,testNamesList)
```

```

def getUsernameAndTestNames(function):
    rootFile=os.path.abspath(os.curdir)
    listOfFiles=os.listdir(rootFile)
    print(listOfFiles)

    testNamesList=[]
    usernamesList=[]

    for i in range(0,len(listOfFiles)):
        file=listOfFiles[i]
        if file[-10:]=="grades.csv":
            testNamesList.append(file[0:-13])

    print(testNamesList)

    if function=="Test":
        nameFeedbackReciever("Test",usernamesList,testNamesList)

    if function=="Specific":
        csvFile=open('login.csv','r+')
        csvReader=csv.reader(csvFile)
        rows=list(csvReader)

        print(rows)

        for i in range(1,len(rows)):
            username=rows[i][0]
            usernamesList.append(username)

        print(usernamesList)
        csvFile.close()

        nameFeedbackReciever("Specific",usernamesList,testNamesList)

```

usernamesList is defined outside of the if statement so that it can be used both function calls.

Moved outside is statement so that the testNamesList variable is always defined, as it is used for both specific user and whole class feedback

usernameList passed in whole test feedback giving function to prevent function call error as 3 arguments are used. However, at this point the array is empty saving memory

testNamesList variable passed in function call

Next, I will change the nameFeedbackReciever function to use the new variable that is passed through the function.  
(before)

```

def nameFeedbackReciever(function,usernamesList,testNamesList):
    nameFeedbackReciever=Tk()
    nameFeedbackReciever.geometry("350x90")
    nameFeedbackReciever.configure(bg="#33BFFF")

    nameFeedbackReciever.resizable(False,False)

    nameOfFeedbackReciever=Entry(nameFeedbackReciever,width=52)
    nameOfFeedbackReciever.place(x=22,y=22)

    if function=="Specific":
        nameFeedbackReciever.title("Username to give feedback to")

        labelFunction=Label(nameFeedbackReciever,text="Enter the name of a user:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
        labelFunction.place(x=22,y=5)

        submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda: checkFeedbackReciever("Specific",nameOfFeedbackReciever,listToPrint))
        submitButton.place(x=270,y=52)

        printButton=Button(nameFeedbackReciever,text="Print all usernames",command=lambda: printList("Specific",usernamesList))
        printButton.place(x=30,y=52)

    elif function=="Test":
        nameFeedbackReciever.title("Test name to give feedback on")
        nameFeedbackReciever.geometry("350x90")

        labelFunction=Label(nameFeedbackReciever,text="Enter the name of a test:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
        labelFunction.place(x=22,y=5)

        submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda: checkFeedbackReciever("Test",nameOfFeedbackReciever,listToPrint))
        submitButton.place(x=270,y=52)

        print1Button=Button(nameFeedbackReciever,text="Print all test names",command=lambda: printList("Test",testNamesList))
        print1Button.place(x=30,y=52)

        print2Button=Button(nameFeedbackReciever,text="Print all usernames",command=lambda: printList("Specific",usernamesList))
        print2Button.place(x=30,y=52)

```

(after)

New parameters have been added

```

def nameFeedbackReciever(function,usernamesList,testNamesList):
    nameFeedbackReciever=Tk()
    nameFeedbackReciever.geometry("350x90")
    nameFeedbackReciever.configure(bg="#33BFFF")

    nameFeedbackReciever.resizable(False,False)

    nameOfFeedbackTest=Entry(nameFeedbackReciever,width=52)
    nameOfFeedbackTest.place(x=22,y=22)

    labellFunction=Label(nameFeedbackReciever,text="Enter the name of the test:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
    labellFunction.place(x=22,y=5)

    print1Button=Button(nameFeedbackReciever,text="Print all test names",command=lambda: printList("Test",testNamesList))
    print1Button.place(x=30,y=52)

if function=="Test":
    submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda: checkFeedbackTest(nameOfFeedbackTest,testNamesList))
    submitButton.place(x=270,y=52)

    nameFeedbackReciever.title("Test name to give feedback on")

```

Default configuration of the GUI, all of these widgets are needed when giving feedback to both a specific user and a whole class feedback

Features specific to giving feedback to a whole class are defined under the if statement

New function assigned to submit button, specific to checking the information for submitting whole class feedback

```

elif function=="Specific":
    nameFeedbackReciever.title("Username to give feedback to")
    nameFeedbackReciever.geometry("350x150")

    nameOfFeedbackUser=Entry(nameFeedbackReciever,width=52)
    nameOfFeedbackUser.place(x=22,y=69)

    label2Function=Label(nameFeedbackReciever,text="Enter the name of a user:",bg="#33BFFF",font=("Courier 10"), borderwidth=0)
    label2Function.place(x=22,y=52)

    submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda: checkFeedbackSpecific(nameOfFeedbackTest,nameOfFeedbackUser,testNamesList,usernamesList))
    submitButton.place(x=270,y=107)

    print1Button.place(x=30,y=107)

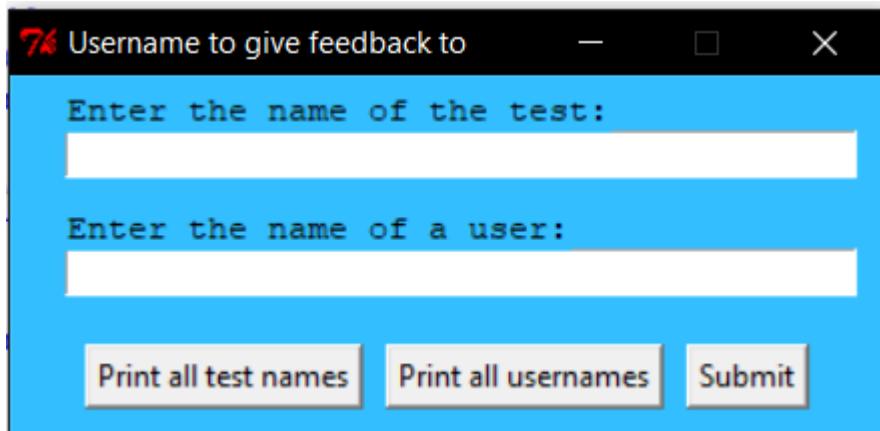
    print2Button=Button(nameFeedbackReciever,text="Print all usernames",command=lambda: printList("Specific",usernamesList))
    print2Button.place(x=150,y=107)

```

Features specific to giving feedback to a specific user are defined under the if statement

New function assigned to submit button, specific to checking the information for submitting specific user feedback

The new specific user feedback GUI displays correctly.



Next, I define the new functions to be used when checking the user inputs. **Otherwise, the user may enter invalid inputs and would not be able to receive an error message, furthermore without validating inputs errors may occur when trying to save the feedback.**

```

def checkFeedbackTest(nameOfFeedbackTest,testNamesList):
    username=""
    testName=nameOfFeedbackTest.get()

    if testName=="":
        messagebox.showerror("Invalid input","Please enter the test name into the input box")
    else:
        for i in range(0,len(testNamesList)):
            if testName==testNamesList[i]:
                giveFeedbackGUI(testName,username)
                break

    elif testName!=testNamesList[i] and i==len(testNamesList)-1:
        messagebox.showerror("Test has not been found","The entered test has not been found, please try again")

```

```

def checkFeedbackSpecific(nameOfFeedbackTest, nameOfFeedbackUser, testNamesList, usernamesList):
    actualUsername=nameOfFeedbackUser.get()
    actualTestName=nameOfFeedbackTest.get()
    username=""
    testName=""

    if actualUsername=="" or actualTestName=="":
        messagebox.showerror("Invalid input","Please make sure both input boxes are filled")

    else:
        for i in range(0,len(testNamesList)):
            if actualTestName==testNamesList[i]:
                testName=actualTestName
                break

        elif actualTestName!=testNamesList[i] and i==len(testNamesList)-1:
            messagebox.showerror("Test has not been found","The entered test has not been found, please try again")

        for j in range(0,len(usernamesList)):
            if actualUsername==usernamesList[j]:
                username=actualUsername
                break

        elif actualUsername!=usernamesList[j] and j==len(usernamesList)-1:
            messagebox.showerror("Username has not been found","The entered username has not been found, please try again")

    if username==actualUsername and testName==actualTestName:
        giveFeedbackGUI(testName,username)

```

Finally, I add the extra parameter to the giveFeedback function and change the parameter names to be more suitable. **Otherwise, I may find it hard to understand what the variables contain when making changes in the future.**

(before)

```
| def giveFeedbackGUI(function,name) :
```

(after)

```
| def giveFeedbackGUI(testName,username) :
```

I also change the function of the button to use the new variables.

(before)

```
submitButton=Button(giveFeedbackGUI,text="Submit",command=lambda: saveFeedback(function,name,feedback))
submitButton.place(x=350,y=215)
```

(after)

```
submitButton=Button(giveFeedbackGUI,text="Submit",command=lambda: saveFeedback(testName,username,feedback))
submitButton.place(x=350,y=215)
```

## Save feedback function

Now I can work on the saveFeedback function, which has yet to be defined.

```
def saveFeedback(testName,username,feedback):
    if username=="":
        username="*"

    csvFile=open(testName,'a')
    newLine=username+feedback
    csvFile.write(newLine)

    csvFile.close()

    messagebox.showinfo("Feedback saved","Feedback has been added successfully")
```

## Error

## Testing

When the program is executed, no errors appear until the “Submit” button is pressed in the giveFeedbackGUI function’s GUI.

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "C:\Users\acer\Desktop\Computing\GUI.py", line 185, in <lambda>
    submitButton=Button(giveFeedbackGUI,text="Submit",command=lambda: saveFeedback(testName,username,feedback))
  File "C:\Users\acer\Desktop\Computing\GUI.py", line 161, in saveFeedback
    newLine=username+feedback
TypeError: Can't convert 'Text' object to str implicitly
```

I realise that feedback is a text widget and not a string variable, so I convert the widget to string. **Otherwise, the feedback cannot be added onto the feedback file and the user will never be able to see their feedback.**

```
def saveFeedback(testName,username,feedback):
    feedback=feedback.get()
```

I also realise that there is no indicator to the user if their feedback is a valid input (not empty), to fix this I add additional if statements to give a message box to the user if their input is not valid. **Otherwise, the user does not know if their feedback has been saved.**

```

def saveFeedback(testName,username,feedback):
    feedback=feedback.get()

    if feedback=="":
        messagebox.showerror("Invalid input","Please enter feedback into the input box")

    elif username=="":
        username="*"

    elif feedback!="":
        csvFile=open(testName,'a')
        newLine=username+feedback
        csvFile.write(newLine)

        csvFile.close()

    messagebox.showinfo("Feedback saved","Feedback has been added successfully")

```

## Error

## Testing

I attempt to test the new function with an empty feedback input; however, I receive a typeError.

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Python32\lib\tkinter\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "C:\Users\acer\Desktop\Computing\GUI.py", line 191, in <lambda>
    submitButton=Button(giveFeedbackGUI,text="Submit",command=lambda: saveFeedback(testName,username,feedback))
  File "C:\Users\acer\Desktop\Computing\GUI.py", line 157, in saveFeedback
    feedback=feedback.get()
TypeError: get() takes at least 2 arguments (1 given)

```

To fix this I define the arguments to get the contents of the widget.

```

def saveFeedback(testName,username,feedback):
    feedback=feedback.get(1.0,'end-1c')

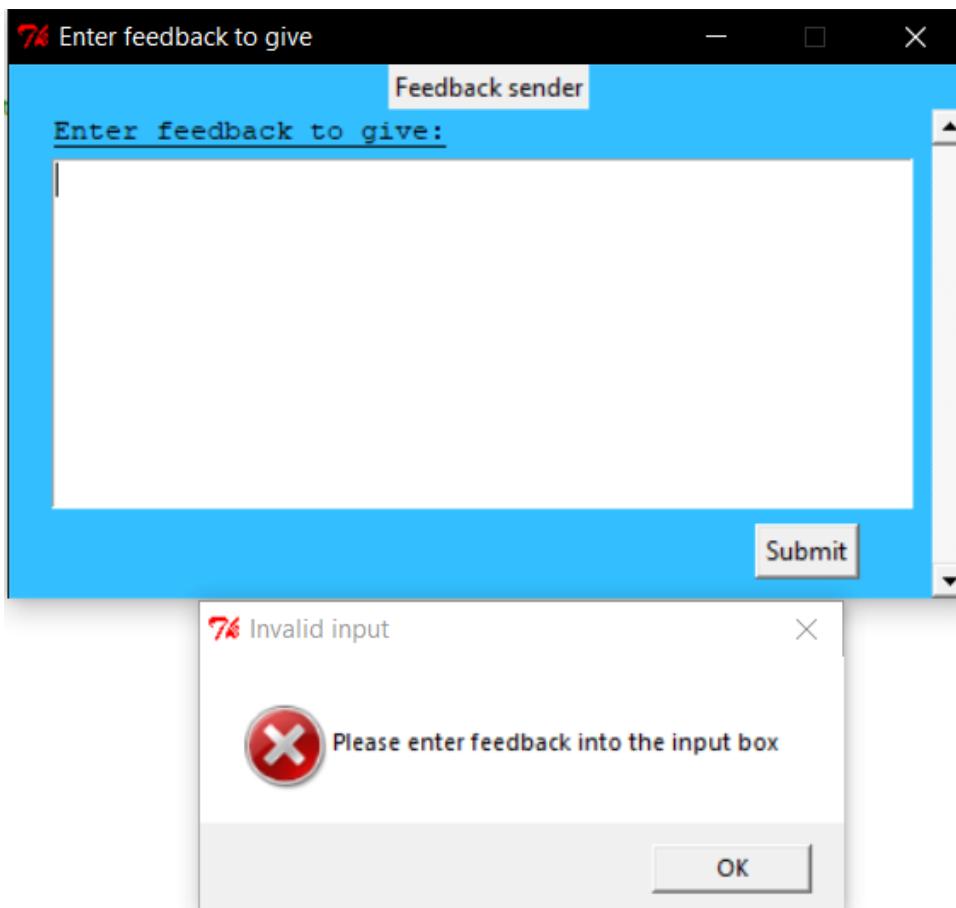
```

Gets all characters in the  
feedback text widget from the  
GUI

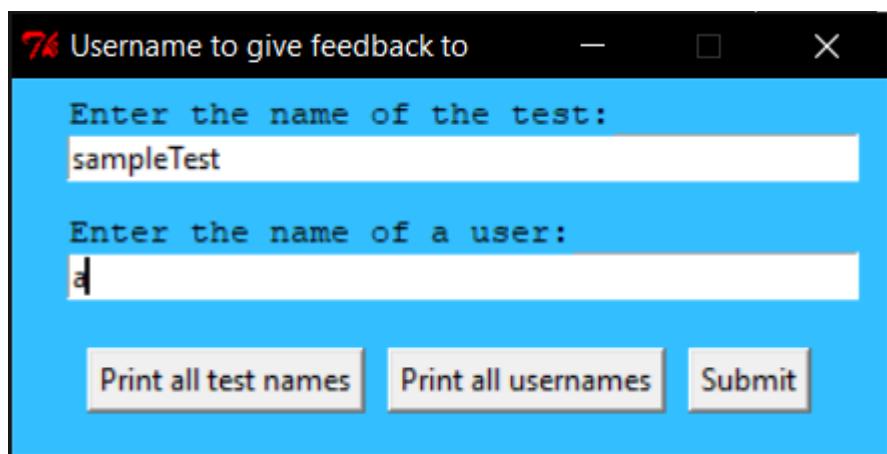
## Testing

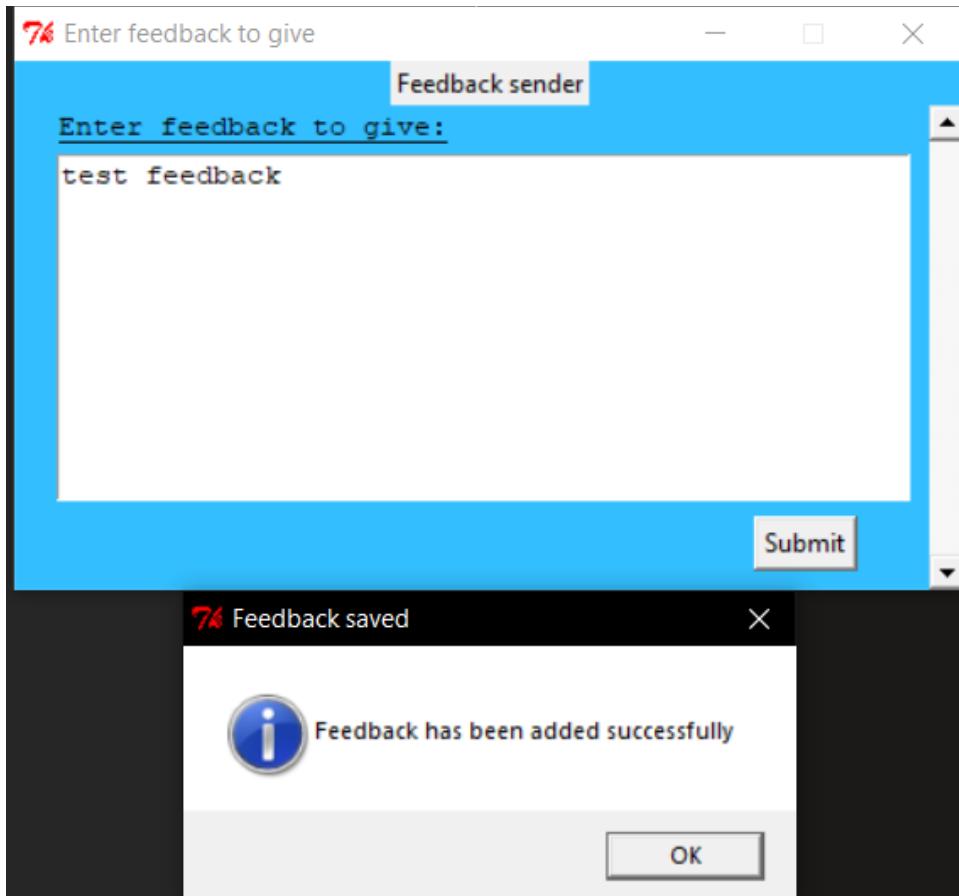
The program now runs with no errors and I test the function.

When the input is empty the error message box appears correctly.



When giving valid feedback to a specific user. The confirmation message box appears correctly, but the feedback is not saved.





	A	B
1	Username	Feedback
2		
3		

I realise that I did not define the test name correctly, so no csv file is modified. I fix this by correcting the testName variable.

(before)

```

def saveFeedback(testName,username,feedback):
    feedback=feedback.get(1.0,'end-1c')

    if feedback=="":
        messagebox.showerror("Invalid input","Please enter feedback into the input box")

    elif username=="":
        username="*"

    elif feedback!="":
        csvFile=open(testName, 'a')
        newLine=username+feedback
        csvFile.write(newLine)

        csvFile.close()

        messagebox.showinfo("Feedback saved","Feedback has been added successfully")

```

(after)

```

def saveFeedback(testName,username,feedback):
    testName=testName+" - feedback.csv" ← The testName variable has the suitable file name
    feedback=feedback.get(1.0,'end-1c')

    if feedback=="":
        messagebox.showerror("Invalid input","Please enter feedback into the input box")

    elif username=="":
        username="*"

    elif feedback!="":
        csvFile=open(testName, 'a')
        newLine=username+feedback
        csvFile.write(newLine)

        csvFile.close()

        messagebox.showinfo("Feedback saved","Feedback has been added successfully")

```

## Testing

Now I test it again with the same test information. When giving valid feedback to a specific user. The confirmation message box appears correctly, but the feedback is saved in the wrong format.

76 Username to give feedback to

Enter the name of the test:  
sampleTest

Enter the name of a user:  
a

76 Enter feedback to give

Feedback sender

Enter feedback to give:  
test feedback

76 Feedback saved

 Feedback has been added successfully

	A	B
1	Username	Feedback
2	atest feedback	

To fix this I add a comma when concatenating the strings. Otherwise, the csv file will not separate the two values.  
(before)

```

elif feedback!="":
    csvFile=open(testName, 'a')
    newLine=username+feedback
    csvFile.write(newLine)

    csvFile.close()

```

(after)

```

elif feedback!="":
    csvFile=open(testName, 'a')
    newLine=username+", "+feedback
    csvFile.write(newLine)

    csvFile.close()

```

Comma added as csv file  
recognises commas as  
column separators.

## Testing

I test the program with the previous test data; however, the feedback is not added properly.

	A	B
1	Username	Feedback
2	atest feedbacka	test feedbacka
3		
4		

To fix this, I will read the entire csv file to a list of lists before adding the username and feedback to that list, then the csv file will be rewritten with the new list and the csv file will be saved.

```

elif feedback!="":
    csvFile=open(testName)
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    rows.append([username,feedback])

    csvFile.close()
    csvFile=open(testName, 'w', newline='')
    csvWriter=csv.writer(csvFile)
    csvWriter.writerows(rows)

    csvFile.close()

    messagebox.showinfo("Feedback saved",

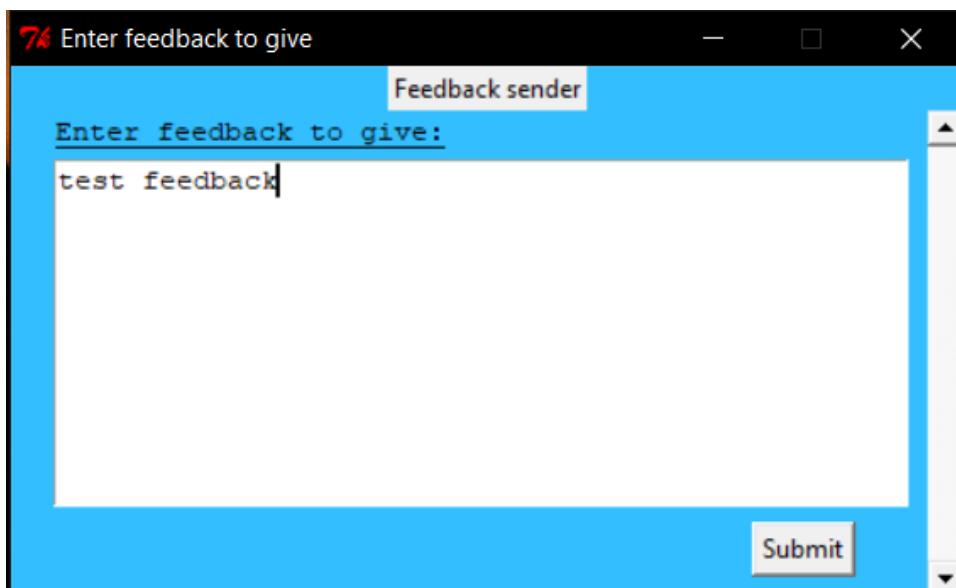
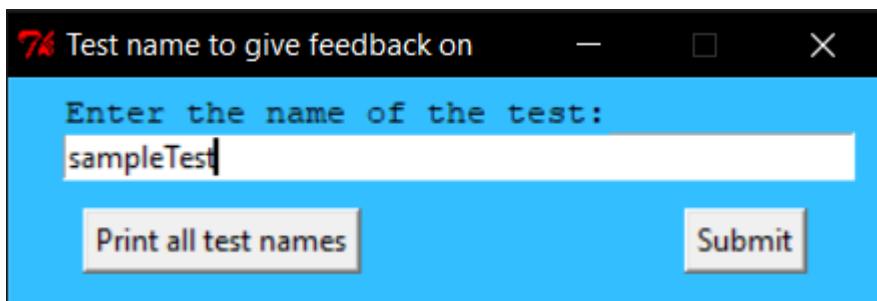
```

## Testing

I test the program with the previous test data and the feedback is saved correctly

	A	B	C
1	Username	Feedback	
2	a	test feedback	
3			
4			

Next, I test the whole class feedback GUI. With the following test data.



However, when the submit button is pressed, nothing happens.

I try to fix this by defining username as \* in an earlier function, as I suspect that passing the empty variable is causing an error.

(before)

```
def checkFeedbackTest(nameOfFeedbackTest,testNamesList):
    username=""
    testName=nameOfFeedbackTest.get()
```

(after)

```
def checkFeedbackTest(nameOfFeedbackTest,testNamesList):
    username="*"
    testName=nameOfFeedbackTest.get()
```

username is defined as "\*" so that an empty variable is not being passed through the functions.

I also remove the if statement in the saveFeedback function that would change the username from being blank to "\*" as the variable is already defined as "\*"

(before)

```
def saveFeedback(testName,username,feedback):
    testName=testName+" - feedback.csv"
    feedback=feedback.get(1.0,'end-lc')

    if feedback=="":
        messagebox.showerror("Invalid input","Please enter feedback into the input box")

    elif username=="":
        username="*"
```

(after)

```
def saveFeedback(testName,username,feedback):
    testName=testName+" - feedback.csv"
    feedback=feedback.get(1.0,'end-lc')

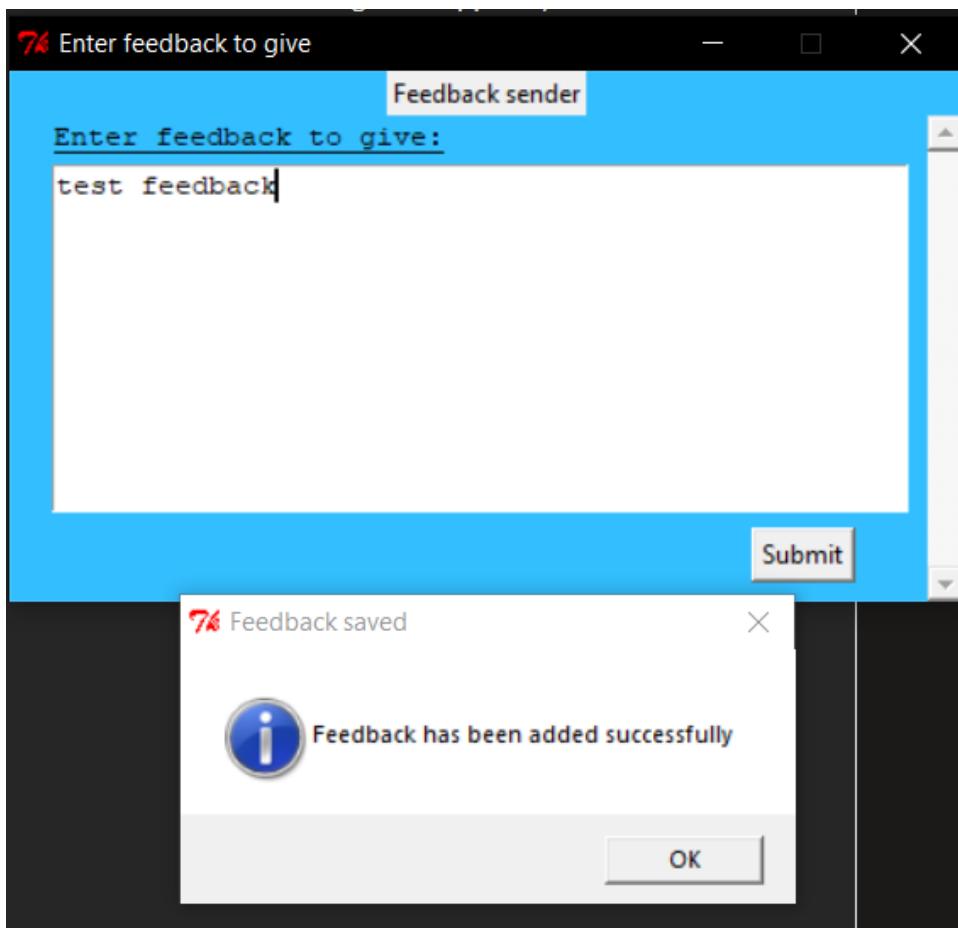
    if feedback=="":
        messagebox.showerror("Invalid input","Please enter feedback into the input box")

    elif feedback!="":
        csvFile=open(testName)
        csvReader=csv.reader(csvFile)
        rows=list(csvReader)
```

if statement has been removed, as it will never be met. **Without removing it, an unnecessary check would be performed, slowing the performance of the program**

## Testing

Now, I try testing again with the previous test data. This time the confirmation message box appears, and the data is saved correctly.



	A	B	C
1	Username	Feedback	
2	a	test feedback	
3	*	test feedback	
4			

This is the end of my first prototype of the feedback sender feature. Next, I will work on my feedback receiver feature.

## Feedback receiver

### Retrieve all feedback function

First, I will need to create a function to get all the feedback designated for a user. I will make a function to search all test marks files and make a list of all the tests that the user has completed. Then, the list of tests will have each of their feedback files searched for feedback either designated for the user or the entire class, all this feedback will be added to a list.

Then, I will make a GUI with a text box that will insert the contents of the list containing all the relevant feedback into the text box making it visible to the user.

I realise my `getUsernamesAndTestNames` function can get all the names of the tests. So, I will make some modifications to this function to make it more suitable for this task.

First, I create a function that I will use to search for the feedback later.

```
def getFeedback(username, listOfFiles):
    return username
```

Next, I need to modify my `getUsernamesAndTestNames` function definition and function call to include a `username` parameter. **Otherwise, the `username` cannot be passed from the secondary menu into the `getFeedback` function, without it becoming a global variable, which would increase the chances of an error.**

```
def getUsernamesAndTestNames(function, username):
    rootFile=os.path.abspath(os.curdir)
    listOfFiles=os.listdir(rootFile)
    print(listOfFiles)
```

Username parameter added to pass username variable from secondary menu

Then I do the same with the function calls in the `feedbackSelectorGUI` function to prevent an error due to the function's parameters and arguments not matching.

```
student", command=lambda: getUsernamesAndTestNames("Specific", username))

, command=lambda: getUsernamesAndTestNames("Test", username))
```

username variable added to prevent function calling error

I also define the `username` variable in the same function (`feedbackSelectorGUI`), to prevent an error due to `username` not being defined despite it being used in the function calls.

```
def feedbackSelectorGUI():
    username="**"

    feedbackSelector=Tk()
    feedbackSelector.geometry("450x160")
    feedbackSelector.title("Feedback sender")
    feedbackSelector.configure(bg="#33BFFF")
```

Username variable defined to allow `getUsernamesAndTestNames` function to run with no errors in all function calls

Then I add an if statement to pass the username and list of test files into the getFeedback function. **Otherwise, the relevant feedback for the logged in user cannot be retrieved.**

```
def getUsersnamesAndTestNames(function,username):
    rootFile=os.path.abspath(os.curdir)
    listOfFiles=os.listdir(rootFile)
    print(listOfFiles)

    testNamesList=[]
    usernamesList=[]

    for i in range(0,len(listOfFiles)):
        file=listOfFiles[i]
        if file[-10:]=="grades.csv":
            testNamesList.append(file[0:-13])

    print(testNamesList)

    if function=="Recieve":
        getFeedback(username,testNamesList)

    if function=="Test":
        nameFeedbackReciever("Test",usernamesList,testNamesList)
```

getFeedback function is called with both required variables, allowing for both to be used in the getFeedback function

Finally, I set the “Receive feedback” button in the secondary menu GUI, to call the getUsersnamesAndTestNames function with the required variables. **Otherwise, I would not know if the receiving feedback system works.** I also add some print statements to test whether the variables are correctly passed to the getFeedback function. **Otherwise, I would not know if the correct variables are being passed.**

```
def secondaryMenu(username,adminStatus):
    sec=Tk()
    sec.geometry("450x50")
    sec.title("Features Menu")
    sec.configure(bg="#33BFFF")

    sec.resizable(False,False)

    labelSecMenu=Label(sec,text="Welcome to the program")
    labelSecMenu.pack(side=TOP)

    buttonComplete=Button(sec,text="Complete a test",command=lambda: nameTestFile("Complete a test","Complete",username))
    buttonComplete.place(x=10, y=20)

    buttonRFeedback=Button(sec,text="Recieve feedback",command=lambda: getUsersnamesAndTestNames("Recieve",username))
    buttonRFeedback.place(x=330,y=20)
```

getUsersnamesAndTestNames function is called with variables that will pass the username function to the getFeedback function

```
def getFeedback(username, listOfFiles):
    print(username)
    print(listOfFiles)
```

Print statements will output the variables that have been passed into the function. **Without this, I will not know if the variables have been correctly passed**

## Testing

When I test the “Receive feedback” button, no error is given, and the correct outputs are given.

```
>>>
['Computing - Analysis.docx', 'Computing - backup', 'Computing - Design.docx', 'Computing - Development.docx', 'Computing - Evaluation.docx', 'empty.csv', 'GUI.py', 'guide.txt', 'login.csv', 'sampleTest - feedback.csv', 'sampleTest - grades.csv', 'sampleTest - marks.csv', 'sampleTest.csv', 'temp.py', 'test.csv', 'test1234 - feedback .csv', 'test1234 - grades.csv', 'test1234 - marks.csv', 'test1234.csv']
['sampleTest', 'test1234']
a
['sampleTest', 'test1234']
```

Correct, name of username used to login

Correct, name of all test files in the current directory

Now, I will add the code to retrieve all relevant feedback. **Otherwise, the feedback would not be helpful to the user.**

```
def getFeedback(username,listOfFiles):
    relevantFiles=[]
    feedback=[]

    for i in range(0,len(listOfFiles)):
        currentFile=listOfFiles[i]+" - marks.csv"
        csvFile=open(currentFile,'r+')
        csvReader=csv.reader(csvFile)
        rows=list(csvReader)

        csvFile.close()

        for j in range(1,len(rows)):
            if username==rows[j][0]:
                relevantFiles.append(listOfFiles[i])
                break

    for i in range(0,len(relevantFiles)):
        currentFile=relevantFiles[i]+" - feedback.csv"
        csvFile=open(currentFile,'r+')
        csvReader=csv.reader(csvFile)
        rows=list(csvReader)

        csvFile.close()

        for j in range(1,len(rows)):
            if username==rows[j][0] or rows[j][0]==***:
                feedback.append(rows[j][1])

    print(relevantFiles)
    print(feedback)
```

Looks through mark files, to see which tests have been completed by the logged in user, and adds these tests to an array (relevantFiles)

Looks through feedback files of tests that the logged in user has completed and adds any feedback designated to the whole class or the logged in user to an array (feedback)

## Testing

I test this using the following test files

SampleTest – marks.csv

Username	Marks	Grades
a	7	U
a	7	B
a	7	B
a	7	B
a	7	B
b	7	B

SampleTest – feedback.csv

Username	Feedback
a	sampleTest, correct name, in marks
*	sampleTest, all correct name
b	sampleTest, wrong name, in marks
c	sampleTest, wrong name, not in marks

Test1234 – marks.csv

Username	Marks	Grades
a	2	A
b	2	A

Test1234 – feedback.csv

Username	Feedback
a	test1234, correct name,in marks
*	test1234, all correct name
b	test1234, wrong name, in marks
c	test1234, wrong name, not in marks

With these test values. Only the feedback with the username "a" or "\*", should be added to the array and outputted.

## Testing

After running the program, all expected values are given.

```
['sampleTest', 'test1234']
['sampleTest, correct name, in marks', 'sampleTest, all correct name', 'test1234
, correct name,in marks', 'test1234, all correct name']
```

## Display feedback GUI

Next, I create a GUI to display all the feedback. **Otherwise, the feedback would not be visible to the user**

```
def displayFeedback(feedback):
    displayFeedback=Tk()
    displayFeedback.geometry("450x250")
    displayFeedback.title("Your feedback")
    displayFeedback.configure(bg="#33BFFF")

    displayFeedback.resizable(False, False)

    labelFeedbackReciever=Label(displayFeedback, text="Feedback reciever")
    labelFeedbackReciever.pack(side=TOP)

    labelFunction=Label(displayFeedback, text="Your feedback:", bg="#33BFFF", font=("Courier 10 underline"), borderwidth=0)
    labelFunction.place(x=20, y=22)

    displayedFeedback=Text(displayFeedback, width=50, height=10, wrap=WORD)
    displayedFeedback.place(x=20, y=44)

    submitButton=Button(displayFeedback, text="Close", command=lambda: displayFeedback.destroy())
    submitButton.place(x=350, y=215)

    scrollbar=Scrollbar(displayFeedback)
    scrollbar.pack(side=RIGHT, fill=Y)

    displayedFeedback.config(yscrollcommand=scrollbar.set)
    scrollbar.config(command=displayedFeedback.yview)

    for i in range(0, len(feedback)):
        currentFeedback=feedback[i]+"\n\n"
        displayedFeedback.insert(END, currentFeedback)
```

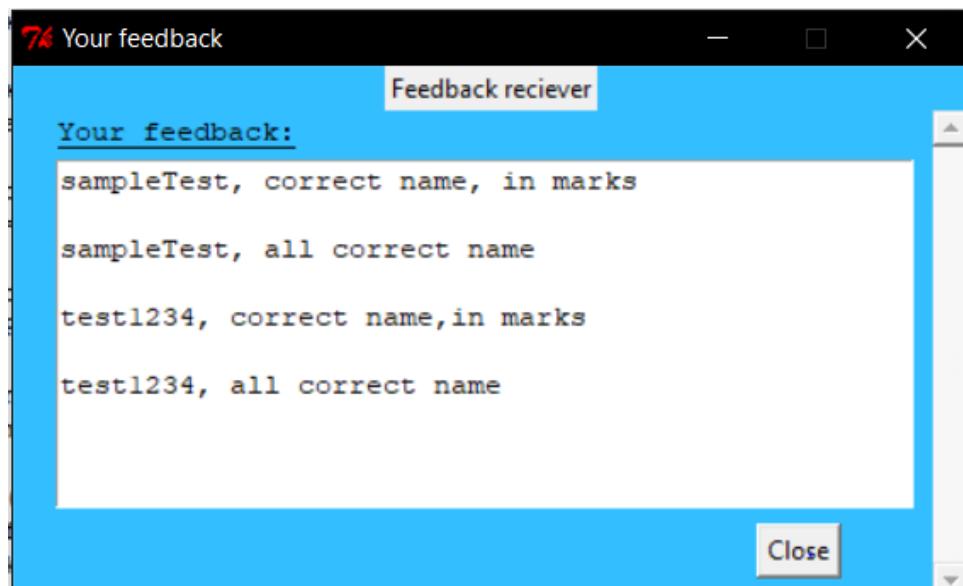
I then add a function call to the getFeedback function to open the GUI with the collected feedback. **Otherwise, the feedback would not be visible to the user**

(in getFeedback function)

```
print(relevantFiles)
print(feedback)

displayFeedback(feedback)
```

When running the program, the GUI appears correctly.



This is the end of my first feedback receiver prototype.

## Prototypes 2

### Registration version 2

During the development of my program along with feedback received from the client, teachers and students, I have made a list of changes that would improve the program.

First, I will implement a system to check if a username is already taken during registration

I add the following code to the registerAccount function

```
csvFile=open('login.csv','r+')
csvReader=csv.reader(csvFile)
rows=list(csvReader)
csvFile.close()

usernameTaken=False

for i in range(1,len(rows)):
    if username==rows[i][0]:
        usernameTaken=True
        break

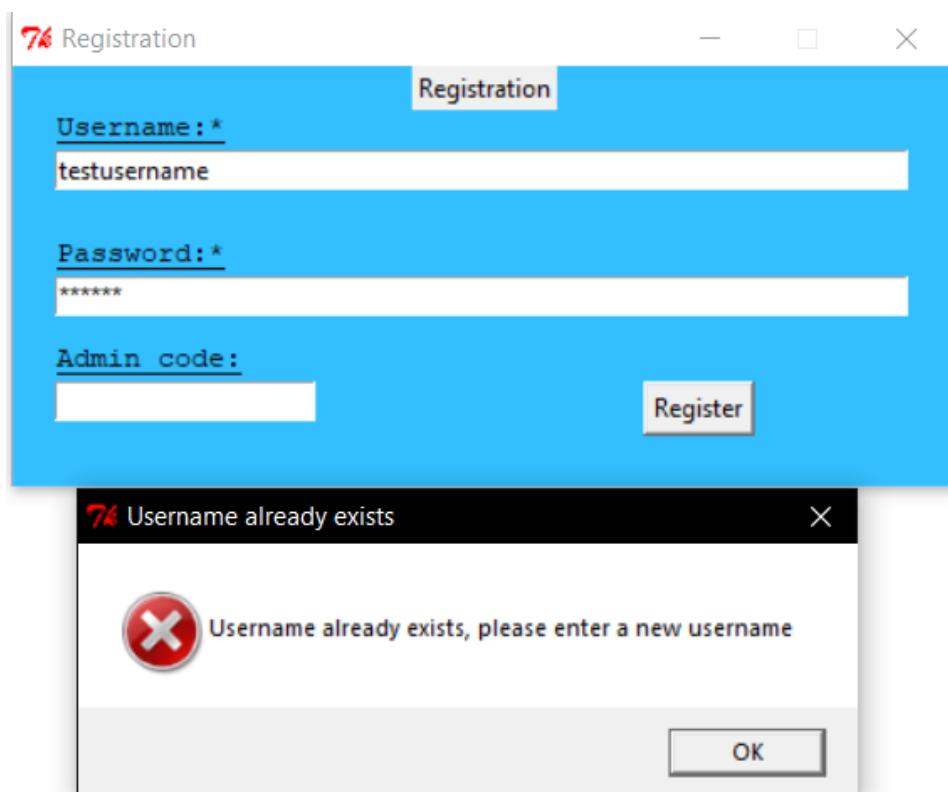
if username!="" and password!="":
    if usernameTaken==False:
        logindata={'Username':username,'Password':password,'Admin':admin}
        login.append(logindata)
        saveLoginData(login)
    elif usernameTaken==True:
        messagebox.showerror("Username already exists","Username already exists, please enter a new username")
    else:
        messagebox.showerror("Registration unsuccessful","Registration unsuccessful, please try again")
```

### Testing

I test the new code against the login.csv file.

	A	B	C
1	Username	Password	Admin
2	testusername	testpassword	testadmin
3	noadmin1	noadmin2	
4	a	a	testadmin
5	input1	input2	input3
6	a		
7	a		

When entering an existing username, such as “testusername”. The correct error message is given.



Next, I will introduce a new colour scheme to the registration system. This was suggested by my client during the post client interview with teachers. **Otherwise, the program would look less aesthetically pleasing.** I will change the colour of each respective system, e.g.: red for registering, yellow for logging in.

(Before)

```
def registerFunction():
    registerWindow=Tk()
    registerWindow.geometry("450x200")
    registerWindow.title("Registration")
    registerWindow.configure(bg="#33BFFF")

    labelRegistration=Label(registerWindow, text="Registration")
    labelRegistration.pack(side=TOP)

    registerWindow.resizable(False, False)

    labelUsername=Label(registerWindow, text="Username:", bg="#33BFFF",
    labelUsername.place(x=20, y=20)

    usernameBox=Entry(registerWindow, width=67)
    usernameBox.place(x=20, y=40)

    labelPassword=Label(registerWindow, text="Password:", bg="#33BFFF",
    labelPassword.place(x=20, y=80)

    passwordBox=Entry(registerWindow, width=67, show="*")
    passwordBox.place(x=20, y=100)

    labelAdmin=Label(registerWindow, text="Admin code:", bg="#33BFFF", for
```



(After)

```
def registerFunction():
    registerWindow=Tk()
    registerWindow.geometry("450x200")
    registerWindow.title("Registration")
    registerWindow.configure(bg="#FF2929")
```

Hex values changed for a different colour

```
labelRegistration=Label(registerWindow, text="Registration")
labelRegistration.pack(side=TOP)

registerWindow.resizable(False, False)

labelUsername=Label(registerWindow, text="Username:", bg="#FF2929",
labelUsername.place(x=20, y=20)

usernameBox=Entry(registerWindow, width=67)
usernameBox.place(x=20, y=40)

labelPassword=Label(registerWindow, text="Password:", bg="#FF2929",
labelPassword.place(x=20, y=80)

passwordBox=Entry(registerWindow, width=67, show="**")
passwordBox.place(x=20, y=100)

labelAdmin=Label(registerWindow, text="Admin code:", bg="#FF2929", for
```



Next, I add some more validations to make the program more secure. Accounts will now require a username and password of at least 8 characters long. **Otherwise, the usernames and passwords will be easier to guess making the program less secure.**

(registerAccount function)

(before)

```

usernameTaken=False

for i in range(1,len(rows)):
    if username==rows[i][0]:
        usernameTaken=True
        break

# Checks if the inputted username is not blank and does not already exist, and checks if the inputted pass
if username!="" and password!="":
    logindata={'Username':username,'Password':password,'Admin':admin}
    login.append(logindata)
    # Saves login details if the inputted login details are valid
    saveLoginData(login)
elif usernameTaken==True:
    messagebox.showerror("Username already exists","Username already exists, please enter a new username")
else:
    messagebox.showerror("Registration unsuccessful","Registration unsuccessful, please try again")

```

(after)

```

usernameTaken=False
usernameValid=False
passwordValid=False

for i in range(1,len(rows)):
    if username==rows[i][0]:
        usernameTaken=True
        break

if len(password)>=int(8):
    passwordValid=True
elif len(username)>=int(8):
    usernameValid=True

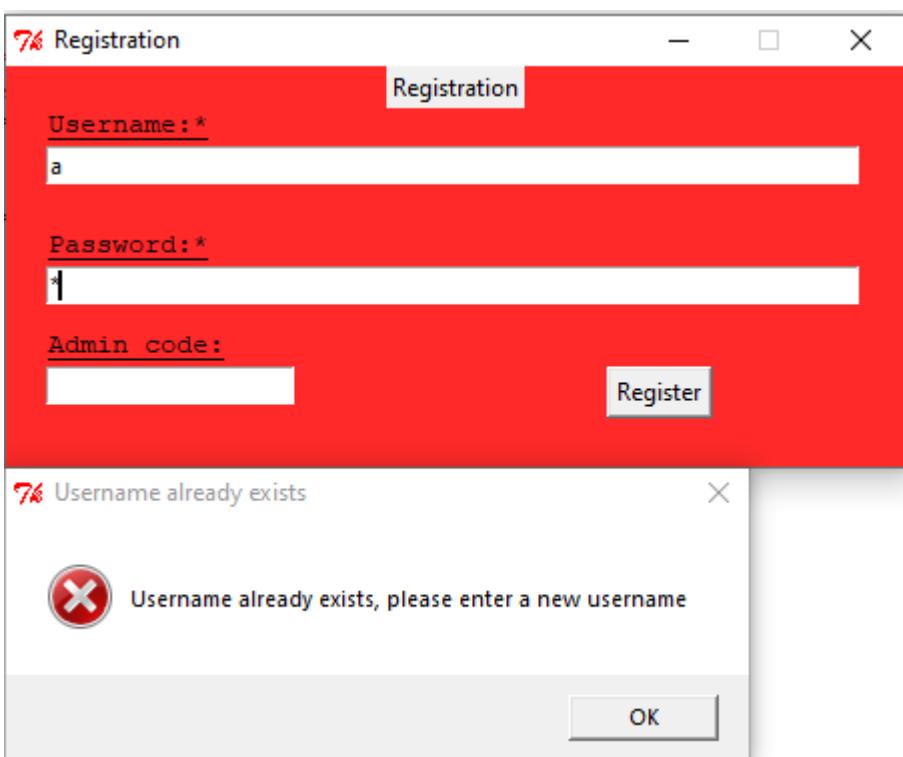
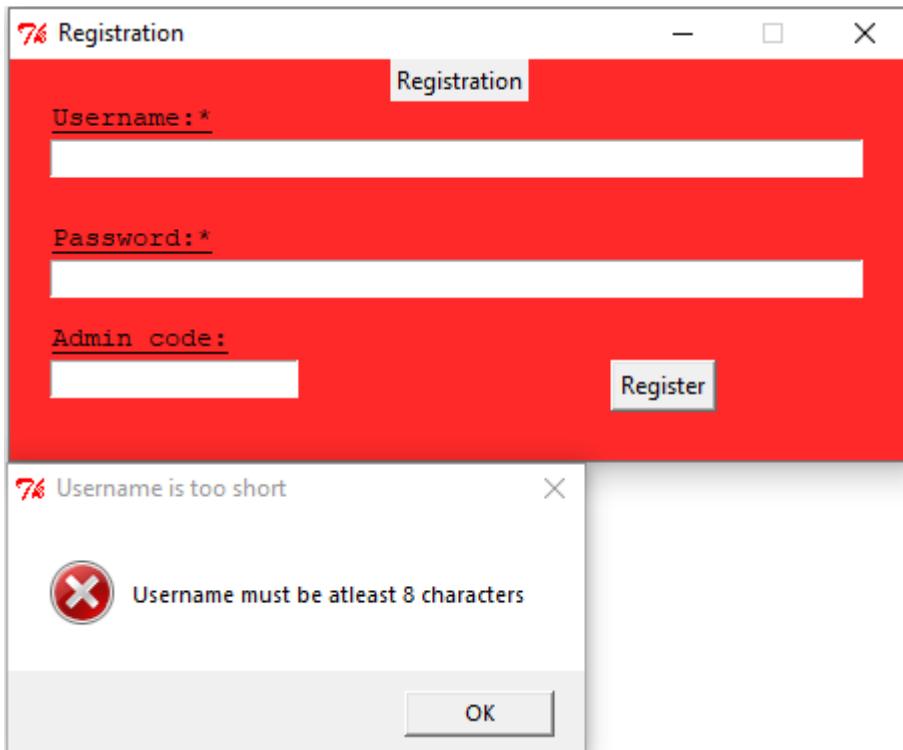
# Checks if the inputted username valid and does not already exist, and checks if the inputted password is
if username!="" and password!="": logindata={'Username':username,'Password':password,'Admin':admin}
login.append(logindata)
# Saves login details if the inputted login details are valid
saveLoginData(login)
elif usernameTaken==True:
    messagebox.showerror("Username already exists","Username already exists, please enter a new username")
elif usernameValid==False:
    messagebox.showerror("Username is too short","Username must be atleast 8 characters")
elif passwordValid==False:
    messagebox.showerror("Password is too short","Password must be atleast 8 characters")
else:
    messagebox.showerror("Registration unsuccessful","Registration unsuccessful, please try again")

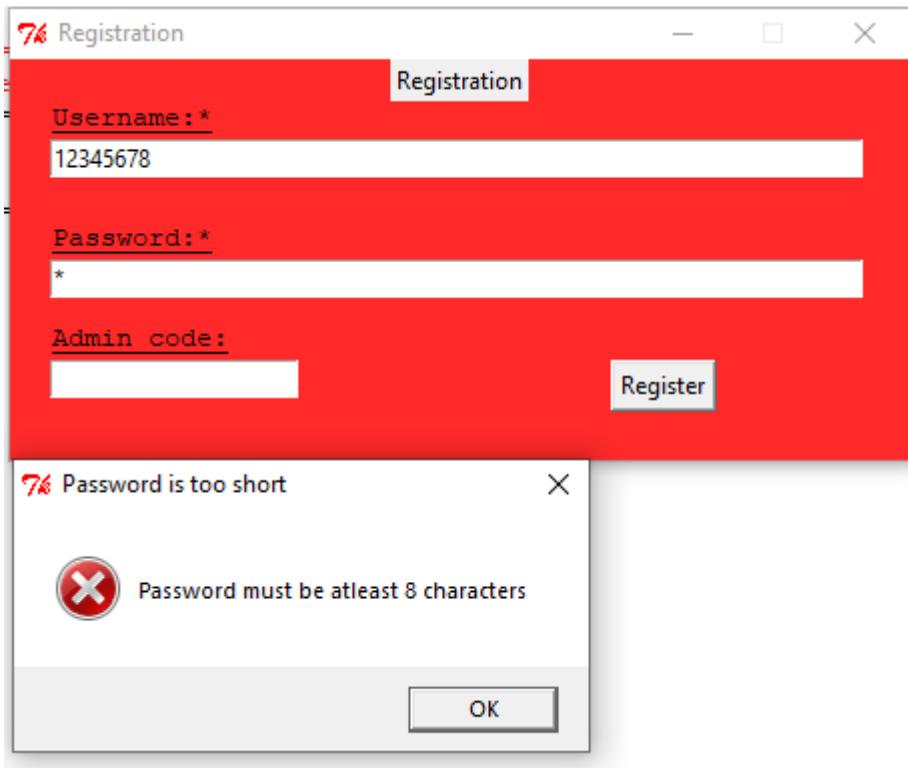
Additional boolean variables added for checks
if statement added to check lengths of user inputs
# Checks if the inputted username valid and does not already exist, and checks if the inputted password is
Additional error message boxes based on failed validations.
Otherwise, the user would not know what is wrong with there inputs

```

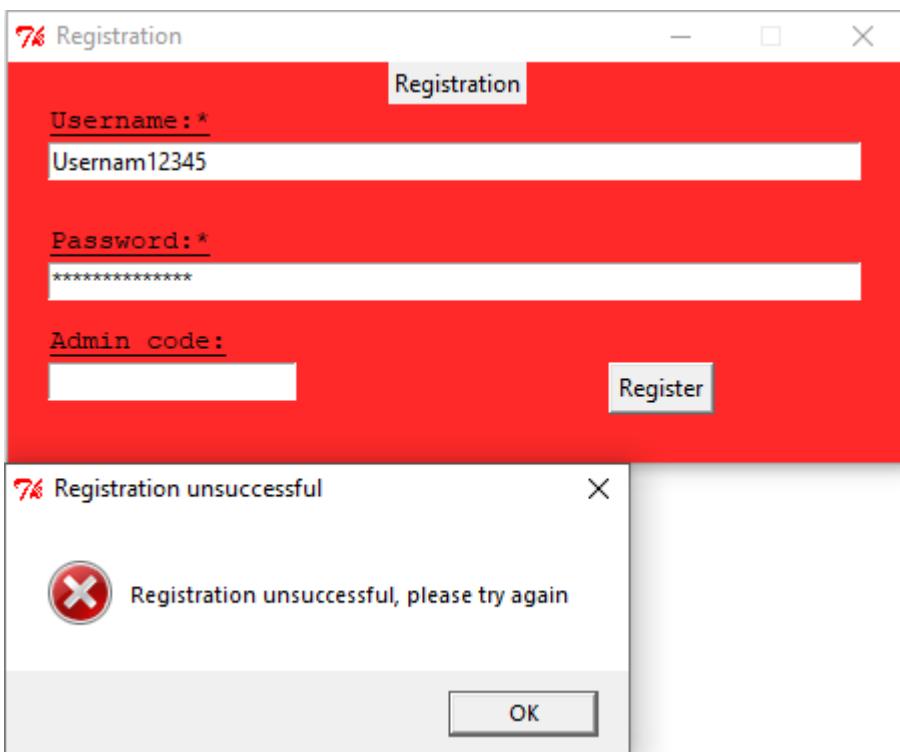
## Testing

I now test the new validations





During testing, I realise that valid details will not be registered.



To fix this I modify the final if statement in the function  
(before)

```

if username!="" and password!="":
    logindata={'Username':username,'Password':password,'Admin':admin}
    login.append(logindata)
    # Saves login details if the inputted login details are valid
    saveLoginData(login)
elif usernameTaken==True:
    messagebox.showerror("Username already exists","Username already exists, please enter a new username")
else:
    if usernameValid==False:
        ...

```

(after)

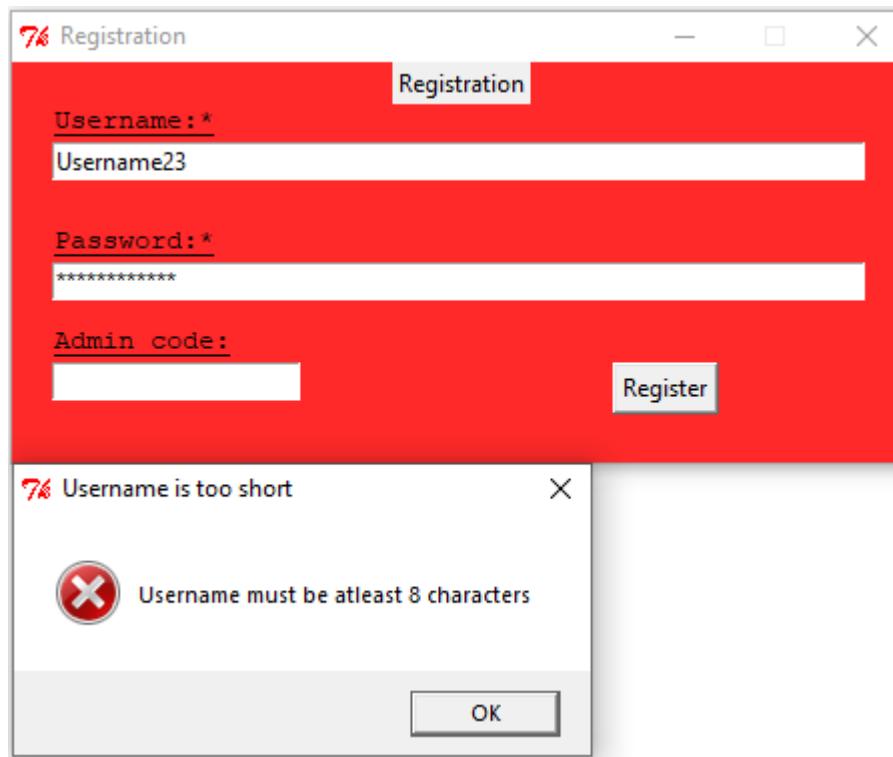
if statement condition changed  
from "usernameTaken==True"  
to "usernameValid==True"

```

if username!="" and password!="":
    logindata={'Username':username,'Password':password,'Admin':admin}
    login.append(logindata)
    # Saves login details if the inputted login details are valid
    saveLoginData(login)
elif usernameTaken==True:
    messagebox.showerror("Username already exists","Username already exists, please enter a new username")
elif usernameValid==False:
    ...

```

Now a different, error message appears



To fix this I modify the if statements that determine whether the username and password is valid or not.

(before)

```

if len(password)>=int(8):
    passwordValid=True
elif len(username)>=int(8):
    usernameValid=True

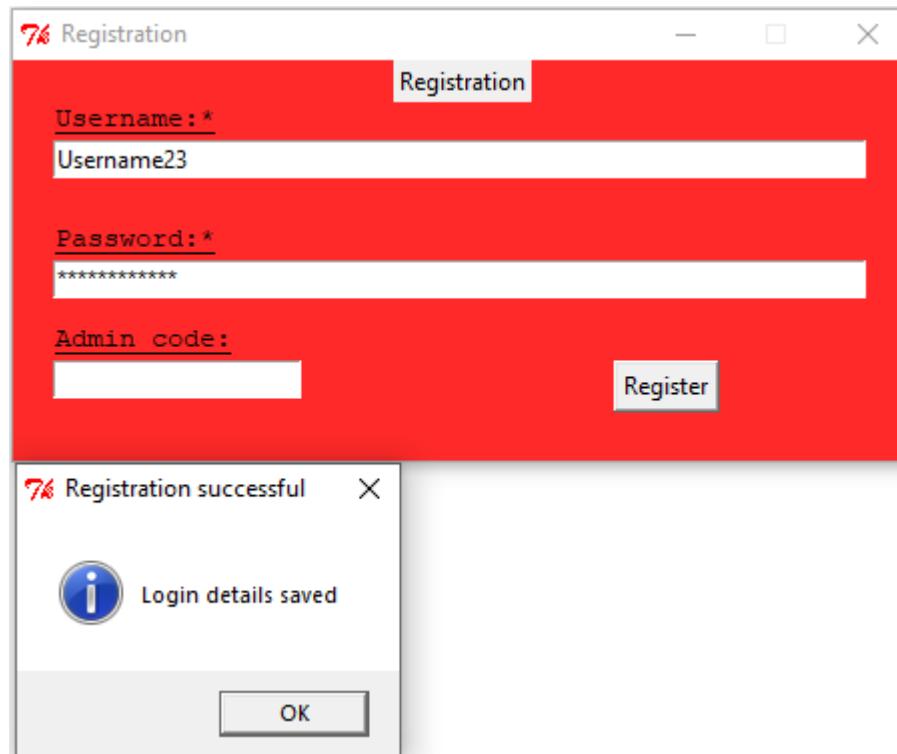
```

(after)

```
if len(password)>=int(8):  
    passwordValid=True|  
  
if len(username)>=int(8):  
    usernameValid=True
```

If statement used so the condition can be checked if the first if statement is met

Now the last output is correct.



Next, I will add the last validation, which is that both the username and password do not start with a number.  
(before)

```

usernameTaken=False
usernameValid=False
passwordValid=False

for i in range(1,len(rows)):
    if username==rows[i][0]:
        usernameTaken=True
        break

if len(password)>=int(8):
    passwordValid=True

if len(username)>=int(8):
    usernameValid=True

# Checks if the inputted username valid and does not already exist, and checks if the inputted password is
if username!="" and password!="":
    if usernameTaken==False and usernameValid==True and passwordValid==True:
        logindata={'Username':username,'Password':password,'Admin':admin}
        login.append(logindata)
        # Saves login details if the inputted login details are valid
        saveLoginData(login)
    elif usernameTaken==True:
        messagebox.showerror("Username already exists","Username already exists, please enter a new username")
    elif usernameValid==False:
        messagebox.showerror("Username is too short","Username must be atleast 8 characters")
    elif passwordValid==False:
        messagebox.showerror("Password is too short","Password must be atleast 8 characters")
    else:
        messagebox.showerror("Registration unsuccessful","Registration unsuccessful, please try again")

```

(after)

```

usernameTaken=False
usernameLength=False
passwordLength=False
userFirstChar=True
passFirstChar=True

for i in range(1,len(rows)):
    if username==rows[i][0]:
        usernameTaken=True
        break

if len(password)>=int(8):
    passwordLength=True

if len(username)>=int(8):
    usernameLength=True

if ord(username[:1])>=int(48) and ord(username[:1])<=int(57):
    userFirstChar=False

if ord(password[0])>=int(48) and ord(password[0])<=int(57):
    passFirstChar=False

if username!="" and password!="" and usernameTaken==False and usernameLength==True and passwordLength==True and userFirstChar==True and passFirstChar==True:
    logindata={'Username':username,'Password':password,'Admin':admin}
    login.append(logindata)

```

New boolean variables added for validations

Variable names changed to more suitable names

Updated variable names used

into if statementAdded new variables

```

logindata={'Username':username,'Password':password,'Admin':admin}
login.append(logindata)
# Saves login details if the inputted login details are valid
saveLoginData(login)
elif usernameTaken==True:
    messagebox.showerror("Username already exists","Username already exists, please enter a new username")
elif usernameLength==False:
    messagebox.showerror("Username is too short","Username must be atleast 8 characters")
elif passwordLength==False:
    messagebox.showerror("Password is too short","Password must be atleast 8 characters")
elif userFirstChar==False:
    messagebox.showerror("Invalid username","Username must not start with a number")
elif passFirstChar==False:
    messagebox.showerror("Invalid password","Password must not start with a number")
else:
    messagebox.showerror("Registration unsuccessful","Registration unsuccessful, please try again")

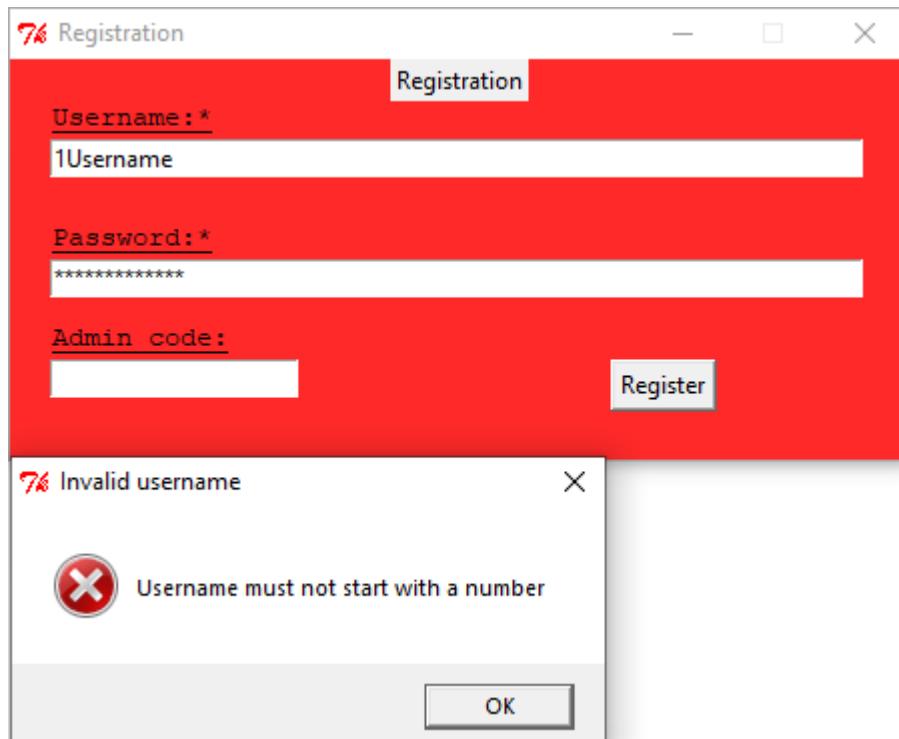
```

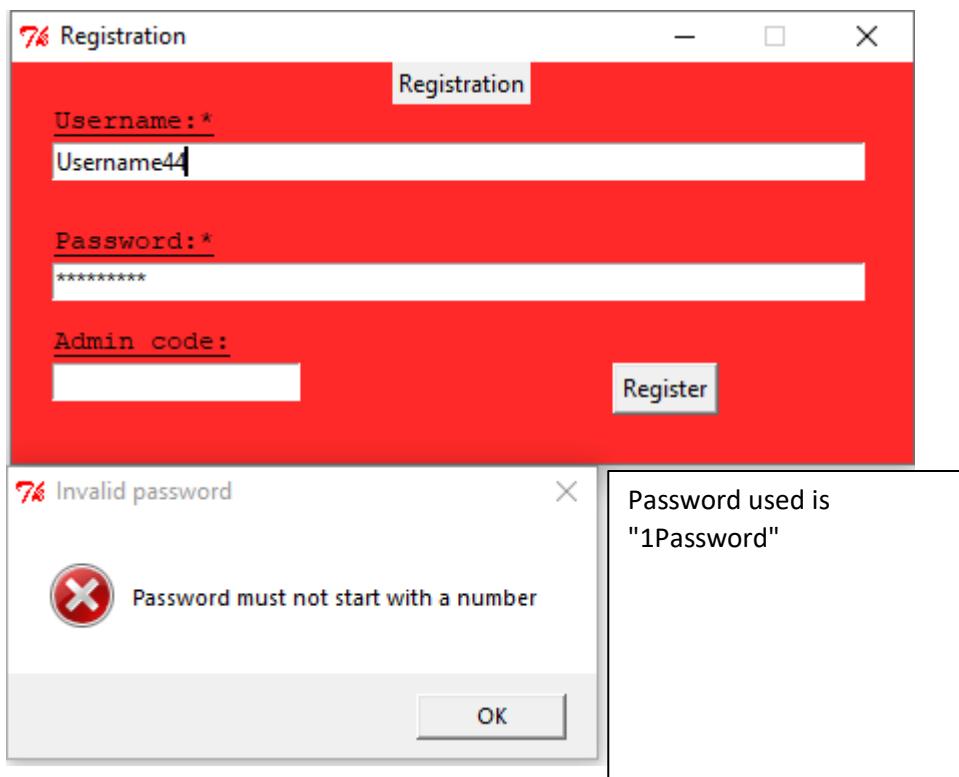
Updated variable names

New variables used to produce relevant error message

## Testing

Then, I test the validations





Both validations work and the correct error messages are given

## Login version 2

During the development of my program along with feedback received from the client, teachers and students, I have made a list of changes that would improve the program.

First, I will introduce a new colour scheme to the login system. This was suggested by my client during the post client interview with teachers. **Otherwise, the program would look less aesthetically pleasing.** I will change the colour of each respective system, e.g.: red for registering, yellow for logging in.

(Before)

```
loginWindow=Tk()
loginWindow.geometry("450x200")
loginWindow.title("Login")
loginWindow.configure(bg="#33BFFF")

labelLogin=Label(loginWindow, text="Login")
labelLogin.pack(side=TOP)

loginWindow.resizable(False, False)

labelUsername=Label(loginWindow, text="Username:", bg="#33BFFF",
labelUsername.place(x=20, y=20)

usernameBox=Entry(loginWindow, width=67)
usernameBox.place(x=20, y=40)

labelPassword=Label(loginWindow, text="Password:", bg="#33BFFF",
```



(After)

```
loginWindow=Tk()
loginWindow.geometry("450x200")
loginWindow.title("Login")
loginWindow.configure(bg="#FFA21F")

labelLogin=Label(loginWindow, text="Login")
labelLogin.pack(side=TOP)

loginWindow.resizable(False, False)

labelUsername=Label(loginWindow, text="Username:", bg="#FFA21F")
labelUsername.place(x=20, y=20)

usernameBox=Entry(loginWindow, width=67)
usernameBox.place(x=20, y=40)

labelPassword=Label(loginWindow, text="Password:", bg="#FFA21F")
```

Hex value changed to  
make a different colour



## Guide version 2

During the development of my program along with feedback received from the client, teachers and students, I have made a list of changes that would improve the program.

First, I will introduce a new colour scheme to the guide. This was suggested by my client during the post client interview with teachers. **Otherwise, the program would look less aesthetically pleasing.** I will change the colour of each respective system, e.g.: red for registering, yellow for logging in.

(Before)

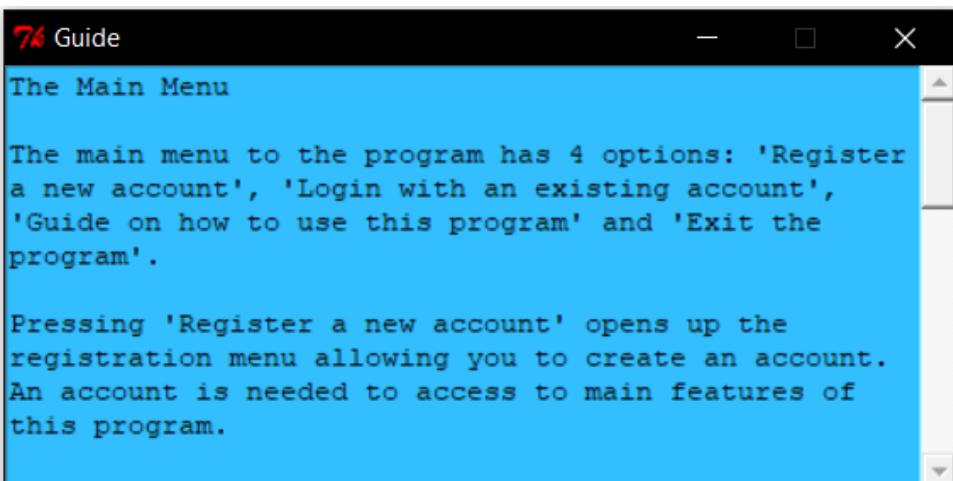
```
guideWindow=Tk()
guideWindow.geometry("450x200")
guideWindow.title("Guide")
guideWindow.configure(bg="#33BFFF")

guideWindow.resizable(False, False)

textFile=open('guide.txt')
text=textFile.read()

scrollbar=Scrollbar(guideWindow)
scrollbar.pack(side=RIGHT, fill=Y)

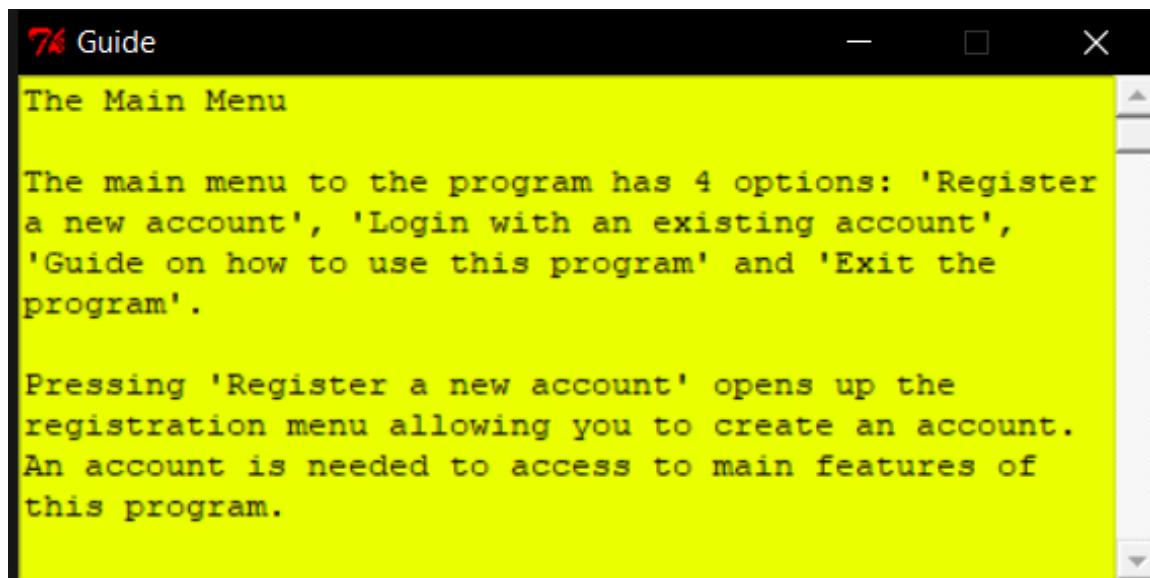
textGuide=Text(guideWindow, bg="#33BFFF",
```



(After)

```
guideWindow=Tk()
guideWindow.geometry("450x200")
guideWindow.title("Guide")
guideWindow.configure(bg="#EAFF00")  
  
guideWindow.resizable(False, False)  
  
textFile=open('guide.txt')
text=textFile.read()  
  
scrollbar=Scrollbar(guideWindow)
scrollbar.pack(side=RIGHT, fill=Y)  
  
textGuide=Text(guideWindow, bg="#EAFF00"
```

Hex values changed to give a different colour



Next, I will update the guide as the guide does not contain information on all the features

(before)

The Main Menu

The main menu to the program has 4 options: 'Register a new account', 'Login with an existing account', 'Guide on how to use this program' and 'Exit the program'.

Pressing 'Register a new account' opens up the registration menu allowing you to create an account. An account is needed to access to main features of this program.

Pressing 'Login with an exisiting account' opens the login menu allowing you to login to an account using existing details.

Pressing 'Guide on how to use this program' opens this text file giving information on the features of this program.

Pressing 'Exit the program' will completely shut down the program.

The Registration Menu

The registration menu allows you to input both a username and a password. These will be used to access the program. You can also input an admin code if it has been provided to you. This determines if you have access to extra features of the program.

The Login Menu

The login menu allows you to input both a username and a password. If exisiting login details have been entered, a secondary menu will open containing the features of the program.

(after, only additional text added has been shown)

#### The Secondary Menu

The secondary menu has 4 options: 'Complete a test', 'Recieve feedback', 'Create a test' and 'Give feedback'. (The last two options are only accessible using an admin account)

Pressing 'Complete a test' starts the complete a test process, allowing a user to test their knowledge using a test.

Pressing 'Recieve feedback' displays all your relevant feedback allowing a user to understand how to improve.

Pressing 'Create a test' starts the create a test process, allowing a user to create a test and add questions and answers to it.

Pressing 'Give feedback' starts the give feedback process, allowing a user to select a user or whole class to give feedback on based on a test's results.

#### Completing a test

First, a window will open where you will need to enter a valid test name. The test file must be in the same directory as the program.

Once a valid test name has been entered, a window containing a question along with an empty answer box will appear. You may enter the answer into the answer input box and press submit when ready. When you finish a test, a window containing your results will open. You can use this to assess how well you did on the test, and you can also attempt the test again by pressing the 'Try again button'.

#### Receiving feedback

A window will appear containing all relevant feedback for you. The feedback given is only for tests that you have attempted. The feedback may be directed specifically to you or to all students who have attempted the test. You can use this feedback to understand how you can improve

#### Creating a test

First, a window will open where you need to enter a valid test name.

Once a valid test name has been entered, a window will open allowing you to create the test. There is a question box and an answers box where you can enter your question and all correct answers. You can enter multiple answers by separating each answer with a comma. When you would like to save the current question and correct answers you can press the 'Submit' button. You can press the 'Preview' button to open a window that shows what the test currently looks like. If you would like to remove the last submitted question and answers, you can press the 'Delete last' button to remove the entries from the test. Once you have finished entering all answers you can press the 'Finish' button to move on to the next process.

Next, a window with several input boxes will open. You need to input numbers for each grade boundary. If you would like to view how many marks the test is out of, you can press the "Check total marks button". When all input boxes have been filled, you can press the 'Submit' button to save the test. The test is now Ready to be completed.

#### Giving feedback

First, a window with two buttons will appear. To give individual feedback, you must press the 'Give feedback to a specific student' button. To give feedback to all students who have completed a test, you must press the 'Give feedback on a whole test'.

If the 'Give feedback to a specific student' button is pressed, a new window will open. In this window you must enter the name of a test and the name a student who has completed the test. You can press either the 'Print all test names' button or the 'Print all usernames' button to view a list of all registered tests and users. Once both input boxes have been filled you can press the 'Submit' button to move on to entering feedback.

If the 'Give feedback on a whole test' button is pressed, a new window will open. In this window you must enter the name of a test. You can press the 'Print all test names' button to view a list of all registered tests. Once the input box has been filled you can press the 'Submit' button to move on to entering feedback.

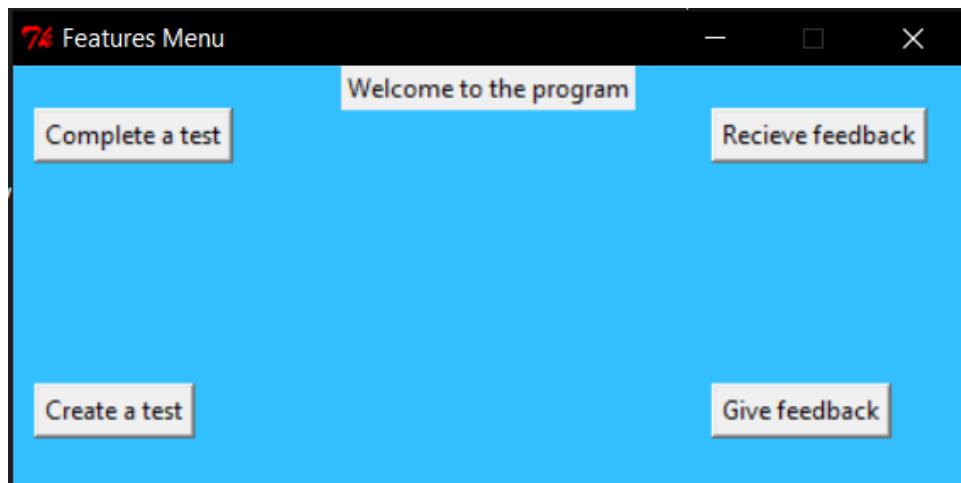
Next, a new window to enter the feedback will open. You can enter your feedback into the input box. You can press 'Submit' once the feedback has been entered to save the feedback

## Secondary menu version 2

First, I will introduce a new colour scheme to the secondary menu. This was suggested by my client during the post client interview with teachers. **Otherwise, the program would look less aesthetically pleasing.** I will change the colour of each respective system, e.g.: red for registering, yellow for logging in.

(before)

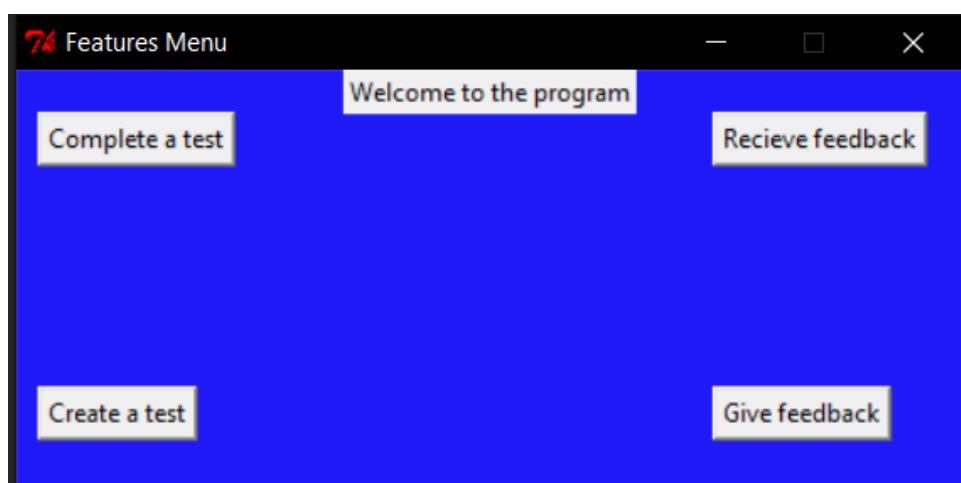
```
def secondaryMenu(username, adminStatus):  
    sec=Tk()  
    sec.geometry("450x50")  
    sec.title("Features Menu")  
    sec.configure(bg="#33BFFF")  
  
    sec.resizable(False, False)
```



(after)

```
def secondaryMenu(username, adminStatus):  
    sec=Tk()  
    sec.geometry("450x50")  
    sec.title("Features Menu")  
    sec.configure(bg="#2019F7")
```

Hex value changed to  
give a different  
colour



## Test creator version 2

During the development of my program along with feedback received from the client, teachers and students, I have made a list of changes that would improve the program.

First, I will implement a check in my testFileCreator function to ensure that a test with a name that already exists cannot be created

(before)

```
def testFileCreator(testName,fileCreator,*gradeAppend):
    csv=".csv"
    userTestName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    dstFile=str(rootFile)+str("\\") +str(userTestName)+str(gradeAppend)
    dstFile=dstFile.strip('()')
    dstFile=dstFile+str(csv)
    rootFile=rootFile+"\empty.csv"
    print(dstFile)

    if userTestName!="":
        shutil.copy2(rootFile,dstFile)

        fileCreator.destroy()

        openCreatedTest(dstFile)

    else:
        messagebox.showerror("Test not created","The test has not been created, please make sure all input boxes are filled and try again")
```

(after)

```
def testFileCreator(testName,fileCreator,*gradeAppend):
    csv=".csv"
    userTestName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    dstFile=str(rootFile)+str("\\") +str(userTestName)+str(gradeAppend)
    dstFile=dstFile.strip('()')
    dstFile=dstFile+str(csv)
    rootFile=rootFile+"\empty.csv"
    print(dstFile)

    nameToCheck=str(userTestName)+str(csv)
    fileExists=False
    listOfFiles=os.listdir(rootFile)

    for i in range(0,len(listOfFiles)):
        file=listOfFiles[i]
        if file==nameToCheck:
            fileExists=True
            break

    if userTestName!="" and fileExists=False:
        shutil.copy2(rootFile,dstFile)

        fileCreator.destroy()

        openCreatedTest(dstFile)

    elif fileExists=True:
        messagebox.showerror("Test not created",
                            "This test already exists, please enter another name")

    else:
        messagebox.showerror("Test not created",
                            "The test has not been created, |please make sure all input boxes are filled and try again")
```

```

def testFileCreator(testName,fileCreator,*gradeAppend):
    csv=".csv"
    userTestName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    dstFile=str(rootFile)+str("\\"+str(userTestName)+str(gradeAppend))
    dstFile=dstFile.strip('()')
    dstFile=dstFile+str(csv)

    nameToCheck=str(userTestName)+str(csv)
    fileExists=False
    listOfFiles=os.listdir(rootFile)

    for i in range(0,len(listOfFiles)):
        file=listOfFiles[i]
        if file==nameToCheck:
            fileExists=True
            break

rootFile=rootFile+"\empty.csv"
print(dstFile)

if userTestName!="" and fileExists==False:
    shutil.copy2(rootFile,dstFile)

    fileCreator.destroy()

    openCreatedTest(dstFile)

elif fileExists==True:
    messagebox.showerror("Test not created",
                         "This test already exists, please enter another name")

else:
    messagebox.showerror("Test not created",
                         "The test has not been created, please make sure all input boxes are filled and try again")

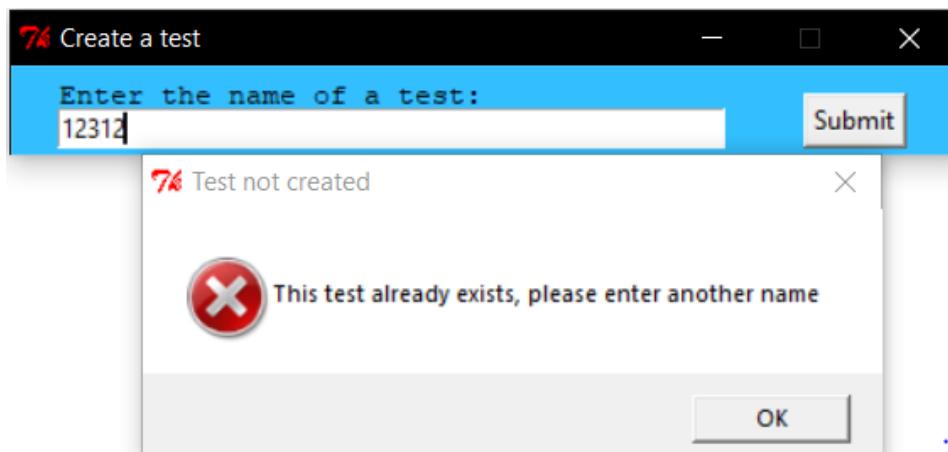
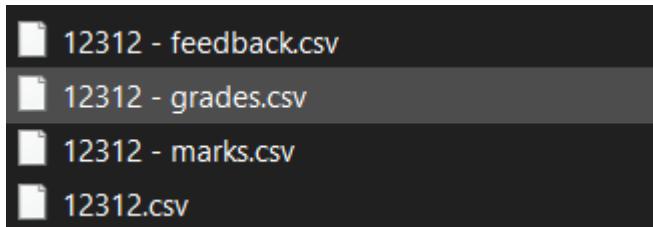
```

Checks if the entered name matches the name of a file in the directory

Only creates the file if the file doesn't already exist

Appropriate error message given if the file already exists. **Otherwise, the user does not know why the test hasn't been created**

I test the statement by entering the name of a test that already exists and I am given the correct error box.



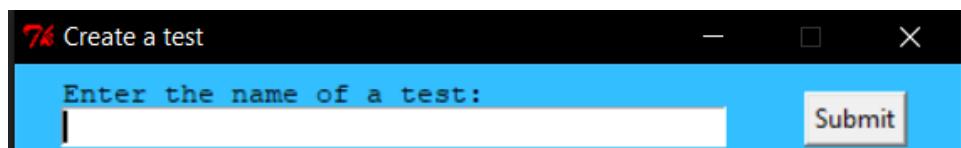
Next, I will introduce a new colour scheme to the test creator GUIs. This was suggested by my client during the post client interview with teachers. **Otherwise, the program would look less aesthetically pleasing.** I will change the colour of each respective system, e.g.: red for registering, yellow for logging in.

(before)

```
def nameTestFile(title,function,username):
    fileCreator=Tk()
    fileCreator.geometry("450x42")
    fileCreator.title(title)
    fileCreator.configure(bg="#33BFFF")

    fileCreator.resizable(False,False)

    labelCreator=Label(fileCreator,text="Enter the name of a test:",bg="#33BFFF",
```



```
def appendTestFile(testFile):
    appendWindow=Tk()
    appendWindow.geometry("450x250")
    appendWindow.title("Create a test")
    appendWindow.configure(bg="#33BFFF")

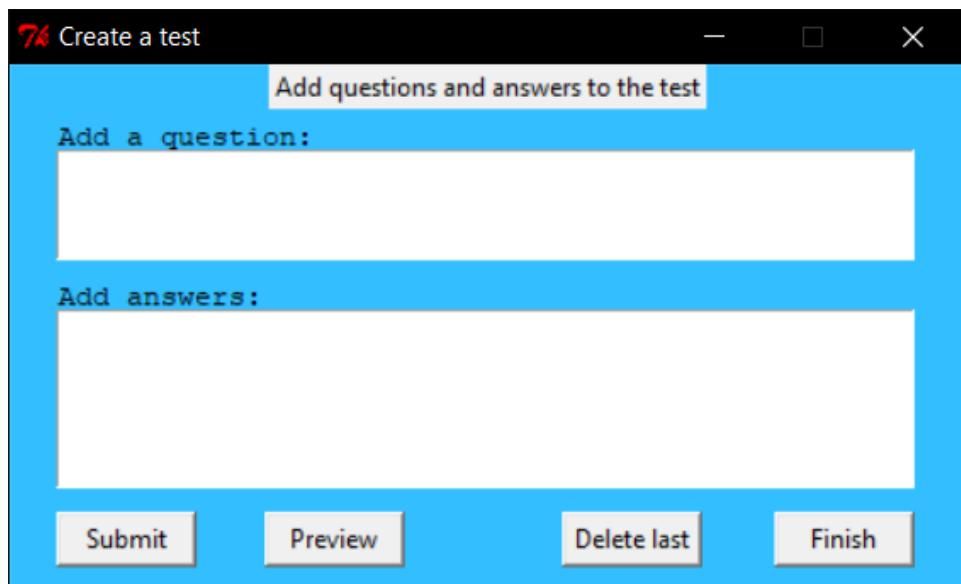
    appendWindow.resizable(False,False)

    labelTitle=Label(appendWindow,text="Add questions and answers to the te:
    labelTitle.pack(side=TOP)

    labelAddQuestion=Label(appendWindow,text="Add a question:",bg="#33BFFF")
    labelAddQuestion.place(x=22,y=25)

    entryAddQuestion=Text(appendWindow,width=50,height=3)
    entryAddQuestion.place(x=22,y=40)

    labelAddAnswer=Label(appendWindow,text="Add answers:",bg="#33BFFF",font:
```

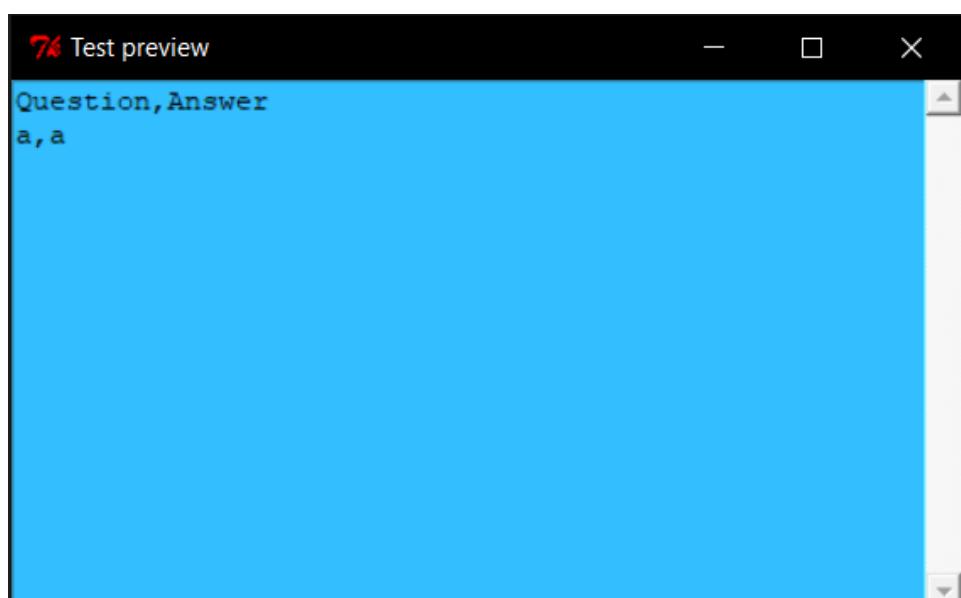


```
def testPreview(file):
    previewWindow=Tk()
    previewWindow.geometry("450x250")
    previewWindow.title("Test preview")
    previewWindow.configure(bg="#33BFFF")

    test=open(file, 'r+')
    testData=test.read()

    scrollbar=Scrollbar(previewWindow)
    scrollbar.pack(side=RIGHT,fill=Y)

    testPreview=Text(previewWindow,bg="#33BFFF"
```



```

def endTest(closeWindow,file):
    gradeWindow=Tk()
    gradeWindow.geometry("300x150")
    gradeWindow.title("Add grade boundaries")
    gradeWindow.configure(bg="#33BFFF")

gradeAlabel=Label(gradeWindow,bg="#33BFFF",text="A:")
gradeAlabel.place(x=20,y=25)

gradeBlabel=Label(gradeWindow,bg="#33BFFF",text="B:")
gradeBlabel.place(x=125,y=25)

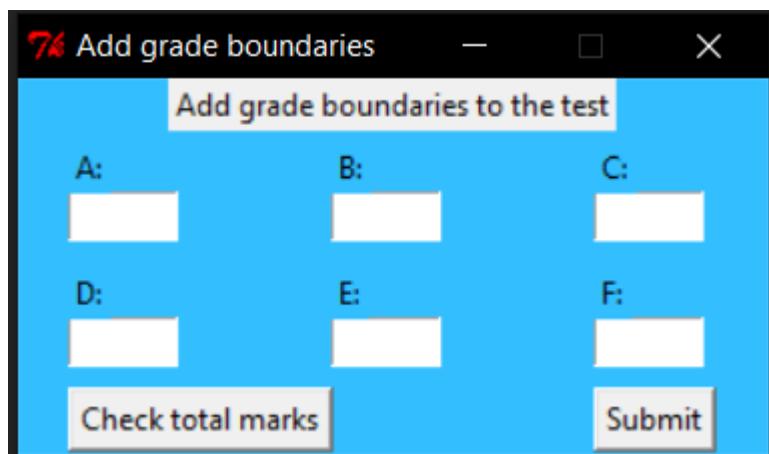
gradeClabel=Label(gradeWindow,bg="#33BFFF",text="C:")
gradeClabel.place(x=230,y=25)

gradeDlabel=Label(gradeWindow,bg="#33BFFF",text="D:")
gradeDlabel.place(x=20,y=75)

gradeElabel=Label(gradeWindow,bg="#33BFFF",text="E:")
gradeElabel.place(x=125,y=75)

gradeFlabel=Label(gradeWindow,bg="#33BFFF",text="F:")
gradeFlabel.place(x=230,y=75)

```



(after)

```

def nameTestFile(title,function,username):
    fileCreator=Tk()
    fileCreator.geometry("450x42")
    fileCreator.title(title)
    fileCreator.configure(bg="#1EFF00")

    fileCreator.resizable(False,False)

    labelCreator=Label(fileCreator,text="Enter the name of a test:",bg="#1EFF00"

```

Hex value changed to give a different colour

Create a test

Enter the name of a test:

Submit

```
def appendTestFile(testFile):
    appendWindow=Tk()
    appendWindow.geometry("450x250")
    appendWindow.title("Create a test")
    appendWindow.configure(bg="#1EFF00")

    appendWindow.resizable(False, False)

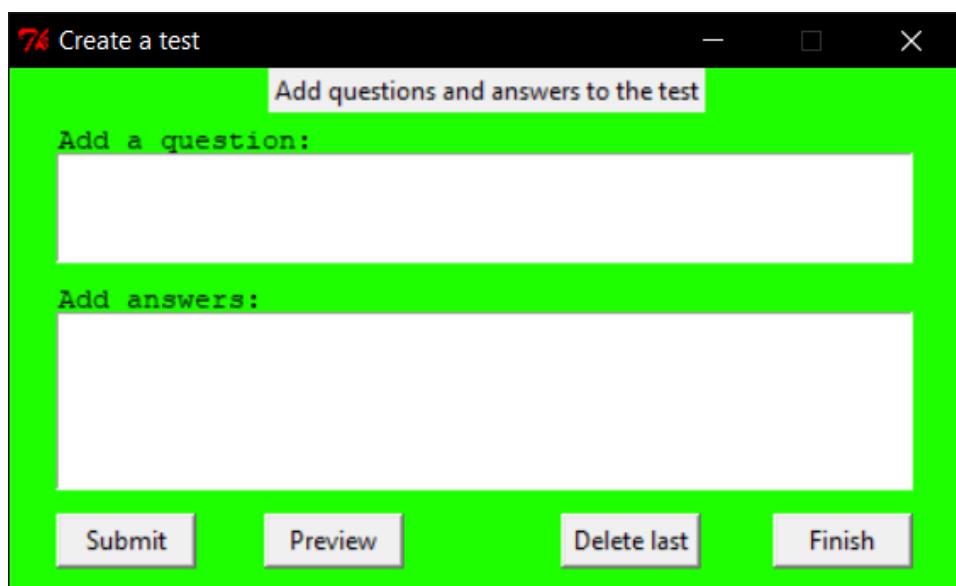
    labelTitle=Label(appendWindow, text="Add questions and answers to the te")
    labelTitle.pack(side=TOP)

    labelAddQuestion=Label(appendWindow, text="Add a question:", bg="#1EFF00")
    labelAddQuestion.place(x=22, y=25)

    entryAddQuestion=Text(appendWindow, width=50, height=3)
    entryAddQuestion.place(x=22, y=40)

    labelAddAnswer=Label(appendWindow, text="Add answers:", bg="#1EFF00", font
```

Hex values changed to give a different colour



```
def testPreview(file):
    previewWindow=Tk()
    previewWindow.geometry("450x250")
    previewWindow.title("Test preview")
    previewWindow.configure(bg="#1EFF00")  
  
    test=open(file,'r+')
    testData=test.read()  
  
    scrollbar=Scrollbar(previewWindow)
    scrollbar.pack(side=RIGHT,fill=Y)  
  
    testPreview=Text(previewWindow,bg="#1EFF00")
```

Hex values changed  
for different colours



```
def endTest(closeWindow,file):
    gradeWindow=Tk()
    gradeWindow.geometry("300x150")
    gradeWindow.title("Add grade boundaries")
    gradeWindow.configure(bg="#1EFF00")
```

Hex values changed for  
different colour

```

gradeAlabel=Label(gradeWindow,bg="#1EFF00",text="A:")
gradeAlabel.place(x=20,y=25)

gradeBlabel=Label(gradeWindow,bg="#1EFF00",text="B:")
gradeBlabel.place(x=125,y=25)

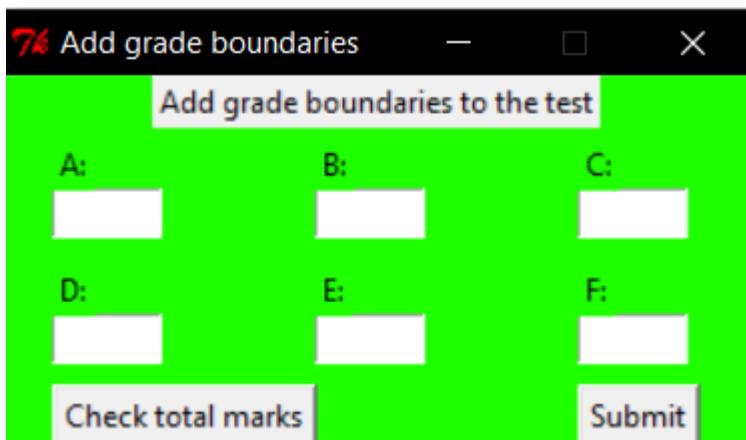
gradeClabel=Label(gradeWindow,bg="#1EFF00",text="C:")
gradeClabel.place(x=230,y=25)

gradeDlabel=Label(gradeWindow,bg="#1EFF00",text="D:")
gradeDlabel.place(x=20,y=75)

gradeElabel=Label(gradeWindow,bg="#1EFF00",text="E:")
gradeElabel.place(x=125,y=75)

gradeFlabel=Label(gradeWindow,bg="#1EFF00",text="F:")

```



When testing my program, I realise that the “Delete last button” during test creation no longer works and an error is given when the button is pressed

```

>>>
N:\\newTest.csv
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\\Python32\\lib\\tkinter\\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\\GUI.py", line 957, in <lambda>
    deleteButton=Button(appendWindow,text="Delete last",width=8,command=lambda:
readTestFile(testFile,"Delete"))
TypeError: readTestFile() takes exactly 3 arguments (2 given)

```

To fix this I add a new argument when calling the readTestFile function to match the amount of parameters the function takes.

```

# Button to delete the last entry in the test file
deleteButton=Button(appendWindow,text="Delete last",width=8,command=lambda: readTestFile(testFile,"Delete"))
deleteButton.place(x=260,y=210)

```

```
# Button to delete the last entry in the test file
deleteButton=Button(appendWindow,text="Delete last",width=8,command=lambda: readTestFile(testFile,"Delete","", ""))
deleteButton.place(x=260,y=210)
```

Empty argument given to match the number of parameters the function takes.  
Otherwise, an error will appear

When creating a new test and pressing the “Delete last” button a new error appears. A similar error has appeared during the development of the program and seems to be based on the hardware used.

```
>>>
N:\\\\test1234.csv
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\\Python32\\lib\\tkinter\\__init__.py", line 1399, in __call__
    return self.func(*args)
  File "N:\\GUI.py", line 957, in <lambda>
    deleteButton=Button(appendWindow,text="Delete last",width=8,command=lambda:
readTestFile(testFile,"Delete","", ""))
  File "N:\\GUI.py", line 369, in readTestFile
    csvFile=open(testFile,'r+')
IOError: [Errno 2] No such file or directory: 'N:\\\\\\\\test1234.csv.csv'
```

Extra backslashes and file extension have been added making the file unreadable by the program

To fix this I add an if statement to change the file name into the correct format, if necessary.

(before)

```
def readTestFile(testName,mode,username):
    # Contents of the test .csv file are read
```

|

```
testFile=testName+".csv"
csvFile=open(testFile,'r+')
csvFileContent=(csvFile)
next(csvFile)
```

(after)

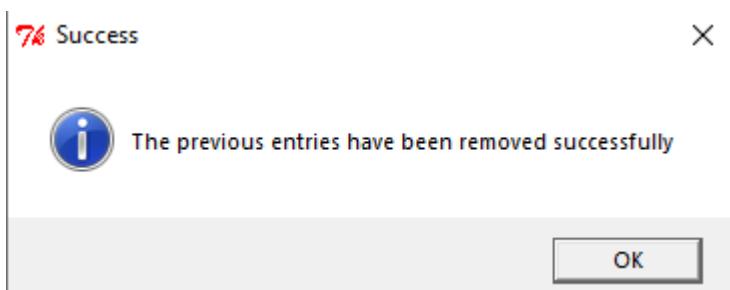
```
def readTestFile(testName, mode, username):
    # Contents of the test .csv file are read

    if testName[-4:] != ".csv":
        testFile=testName+".csv"
    else:
        testFile=testName

    csvFile=open(testFile, 'r+')
    csvFileContent=(csvFile)
    next(csvFile)
```

If the testName variable already ends in ".csv", then the file extension will not be added again

Now the program runs with no errors. However, I realise that when the last entry is removed, it also replaces all other entries with the entry that should be deleted.





To fix this I create a separate function to delete the last entry

```
# Deletes last entry when creating a test
def deleteLast(testFile):
    csvFile=open(testFile, 'r+')
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)
    csvFile.close()

    if len(rows)!=1:
        rows=rows[:-1]

    csvFile=open(testFile, 'w', newline='')
    csvWriter=csv.writer(csvFile)
    csvWriter.writerows(rows)

    csvFile.close()

    messagebox.showinfo("Entry deleted","Last question and answers have been deleted")
else:
    messagebox.showerror("Entry not deleted","There are no entries in the test")
```

I link it to the button

```
# Button to delete the last entry in the test file
deleteButton=Button	appendWindow, text="Delete last", width=8, command=lambda: deleteLast(testFile))
deleteButton.place(x=260, y=210)
```

Button now executes the new function

Next, I remove the unnecessary parts various functions that use either the readTestFile or saveTestData functions  
(before)

```
# Reads the test csv file (also used to delete the last row of a test during the test creation process)
def readTestFile(testName, mode, username):
    # Contents of the test .csv file are read

    if testName[-4:] != ".csv":
        testFile=testName+".csv"
    else:
        testFile=testName

    csvFile=open(testFile, 'r+')
    csvFileContent=(csvFile)
    next(csvFile)

    csvList=[]

    # Contents of the .csv file are turned into a dictionary format
    for row in csvFileContent:
        csv=row.strip().split(",")
        heading=['Question', 'Answer']
        data=zip(heading,csv)
        DataDict=dict(data)
        csvList.append(DataDict)

    print(DataDict)
    print(csvList)
    print(testFile)

    # Deletes last row of the test (test creation)
    if mode=="Delete":
        if len(csvList)!=0:
            csvList.pop()
            saveTestData(csvList,testFile,DataDict,'w+')
        else:
            messagebox.showerror("File is empty","There are no more entries to delete")

    # Reads all the questions and answers
    elif mode=="Read":
        readAllQuestions(testName,testFile,username)
```

(after)

```
# Reads the test csv file
def readTestFile(testName,username):
    # Contents of the test .csv file are read

    if testName[-4:] != ".csv":
        testFile=testName+".csv"
    else:
        testFile=testName

    readAllQuestions(testName,testFile,username)
```

(before)

```
# Checks that the test file exists
def checkFileName(testName,username):
    testName=testName.get()
    # Finds the current directory and adds the test name to the file path
    rootFile=os.path.abspath(os.curdir)
    file=rootFile+"\\"+testName+".csv"

    # Checks if there is a path to the file to see if it exists
    if os.path.isfile(file)==True:
        readTestFile(testName,"Read",username)
    elif os.path.isfile(file)==False:
        messagebox.showerror("File not found","The file has not been found, please try again")
```

(after)

```
# Checks that the test file exists
def checkFileName(testName,username):
    testName=testName.get()
    # Finds the current directory and adds the test name to the file path
    rootFile=os.path.abspath(os.curdir)
    file=rootFile+"\\"+testName+".csv"

    # Checks if there is a path to the file to see if it exists
    if os.path.isfile(file)==True:
        readTestFile(testName,username)
    elif os.path.isfile(file)==False:
        messagebox.showerror("File not found","The file has not been found, please try again")
```

Unnecessary parameter removed

(resultsGUI function)

(before)

```
# Button to start the test completion process again with the same test
tryAgainButton=Button(results,text="Try again",command=lambda: readTestFile(testName,"Read",username))
tryAgainButton.place(x=10,y=100)
```

(after)

```
# Button to start the test completion process again with the same test
tryAgainButton=Button(results,text="Try again",command=lambda: readTestFile(testName,username))
tryAgainButton.place(x=10,y=100)
```

Unnecessary parameter removed

(before)

```
# Saves inputted questions and answers to the csv file
def saveTestData(csv,file,csvData,mode):
    try:
        fileHandle=open(file,mode)
        fileContent=fileHandle.read()

        # If the file is empty, the necessary headings are added
        if fileContent.strip()=='':
            fileHandle.write('Question,Answer\n')

        # Adds the questions and answers from the dictionary
        for item in csv:
            fileHandle.write('{Question},{Answer}'.format(**csvData))
            fileHandle.write('\n')

        # Displays relevant message box on success based on the arguments used in the function call
        if mode=='r+':
            messagebox.showinfo("Success","Question and answers added successfully")
        elif mode=='w+':
            messagebox.showinfo("Success","The previous entries have been removed successfully")

        fileHandle.close()
    except OSError:
        print('Can\'t write to file')
```

(after)

```
# Saves inputted questions and answers to the csv file
def saveTestData(csv,file,csvData):
    try:
        fileHandle=open(file,'r+')
        fileContent=fileHandle.read()

        # If the file is empty, the necessary headings are added
        if fileContent.strip()=='':
            fileHandle.write('Question,Answer\n')

        # Adds the questions and answers from the dictionary
        for item in csv:
            fileHandle.write('{Question},{Answer}'.format(**csvData))
            fileHandle.write('\n')

        # Displays message box on success
        messagebox.showinfo("Success","Question and answers added successfully")

        fileHandle.close()
    except OSError:
        print('Can\'t write to file')
```

'mode' parameter removed and  
file is opened in 'r+' mode

Only 1 message box can now  
be displayed

(appendDetails function)

(before)

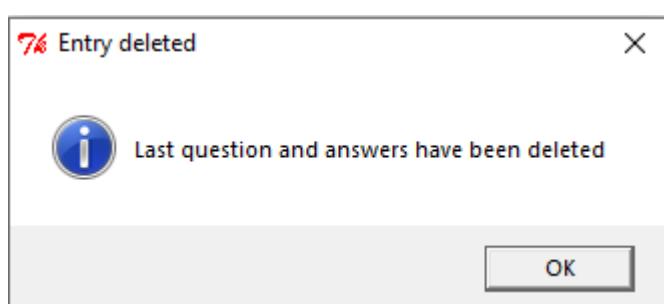
```
if len(question)!=0 and len(answers)!=0:  
    saveTestData(csv,testFile,csvData,'r+')  
else:  
    messagebox.showerror("Questions and ans  
s")
```

(after)

```
if len(question)!=0 and len(answers)!=0:  
    saveTestData(csv,testFile,csvData)  
else:  
    messagebox.showerror("Questions and ans  
s")
```

Unnecessary argument removed from  
function call. **Otherwise, an error would  
appear**

When running the program, where I both created a test and completed a test, no errors were received. The delete last button now works correctly





Next, I will add validations to the grades adding system. **Otherwise, the grade boundaries can take any integer and could overlap**

(before)

```
# Gets inputted grades from GUI
def getGrades(A,B,C,D,E,F,file,gradeWindow):
    # Gets all the user's grades inputs and adds them to a dictionary
    grades=[]
    a=A.get(1.0,'end-1c')
    b=B.get(1.0,'end-1c')
    c=C.get(1.0,'end-1c')
    d=D.get(1.0,'end-1c')
    e=E.get(1.0,'end-1c')
    f=F.get(1.0,'end-1c')
    gradesData={'A':a,'B':b,'C':c,'D':d,'E':e,'F':f}
    grades.append(gradesData)

    # Checks if the user has entered inputs into all input boxes
    if len(a)!=0 and len(b)!=0 and len(c)!=0 and len(d)!=0 and len(e)!=0 and len(f)!=0:
        saveGrades(grades,file,gradesData,gradeWindow)
    else:
        messagebox.showerror("Grades not added","Please fill all input boxes and try again")
```

(after)

```

# Gets inputted grades from GUI
def getGrades(A,B,C,D,E,F,file,gradeWindow):
    # Gets all the user's grades inputs and adds them to a dictionary
    grades=[]
    a=A.get(1.0,'end-1c')
    b=B.get(1.0,'end-1c')
    c=C.get(1.0,'end-1c')
    d=D.get(1.0,'end-1c')
    e=E.get(1.0,'end-1c')
    f=F.get(1.0,'end-1c')
    gradesData={'A':a,'B':b,'C':c,'D':d,'E':e,'F':f}
    grades.append(gradesData)

    # Checks if the user has entered inputs into all input boxes and ensures grade boundaries do not overlap
    if len(a)!=0 and len(b)!=0 and len(c)!=0 and len(d)!=0 and len(e)!=0 and len(f)!=0:
        if a>=b>=c>=d>=e>=f:
            saveGrades(grades,file,gradesData,gradeWindow)
        else:
            messagebox.showerror("Grades not added","Please ensure the grade boundaries do not overlap")
    else:
        messagebox.showerror("Grades not added","Please fill all input boxes and try again")

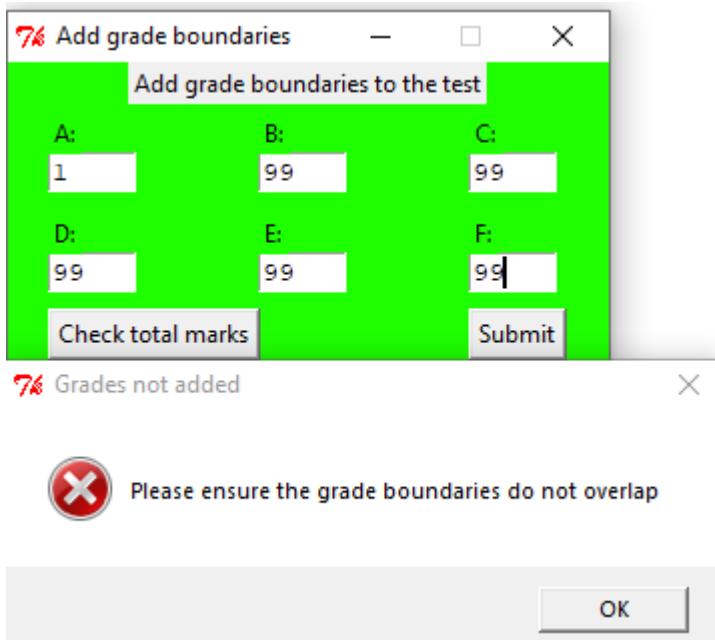
```

New if statement to check if the grade boundaries overlap incorrectly

New error message give if grade boundaries overlap incorrectly. **Otherwise, the user would not know why there inputs are not being accepted**

## Testing

Then, I test the validation with incorrect inputs



The correct error message appears, showing that the validation works

## Test completer version 2

During the development of my program along with feedback received from the client, teachers and students, I have made a list of changes that would improve the program.

First, I will implement a system to delete the contents of the answer box after an answer has been submitted. This makes completing a test more efficient as the user does not have to manually erase the answer box.

(before)

```
def updateQuestionBox(questionBox,questions,answerBox,answers,testName,rowOfEntry,rows):
    answer=answerBox.get(1.0,'end-1c')
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0,'end-1c')

    questionBox.delete(1.0,END)

    i=0

    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            checkAnswer(answer,answers,i,testName,rowOfEntry,rows)
            if i+1<len(questions):
                newQuestion=questions[i+1]
                questionBox.insert(END,newQuestion)
                break
            break

    if i+1==len(questions):
        updateGrade(testName,rowOfEntry)

    if answer=="":
        messagebox.showerror("Error","Please enter your answer into the answer box")
        questionBox.delete(1.0,END)
        questionBox.insert(END,questions[i])

    questionBox.config(state=DISABLED)
```

(after)

```
def updateQuestionBox(questionBox,questions,answerBox,answers,testName,rowOfEntry,rows):
    answer=answerBox.get(1.0,'end-1c')
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0,'end-1c')

    questionBox.delete(1.0,END)

    i=0

    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            checkAnswer(answer,answers,i,testName,rowOfEntry,rows)
            if i+1<len(questions):
                newQuestion=questions[i+1]
                questionBox.insert(END,newQuestion)
                break
            break

    if i+1==len(questions):
        updateGrade(testName,rowOfEntry)

    if answer=="":
        messagebox.showerror("Error","Please enter your answer into the answer box")
        questionBox.delete(1.0,END)
        questionBox.insert(END,questions[i])

    answerBox.delete(1.0,END)                         Deletes the contents of the
    questionBox.config(state=DISABLED)                  answer box
```

Next, I will introduce a new colour scheme to the test completion GUIs. This was suggested by my client during the post client interview with teachers. **Otherwise, the program would look less aesthetically pleasing.** I will change the colour of each respective system, e.g.: red for registering, yellow for logging in.

(before)

```
def testCompletion(testName,questions,answers,rowOfEntry,rows):
    completeWindow=Tk()
    completeWindow.geometry("450x250")
    completeWindow.title(testName)
    completeWindow.configure(bg="#33BFFF")

    completeWindow.resizable(False,False)

    scrollbar=Scrollbar(completeWindow)
    scrollbar.pack(side=RIGHT,fill=Y)

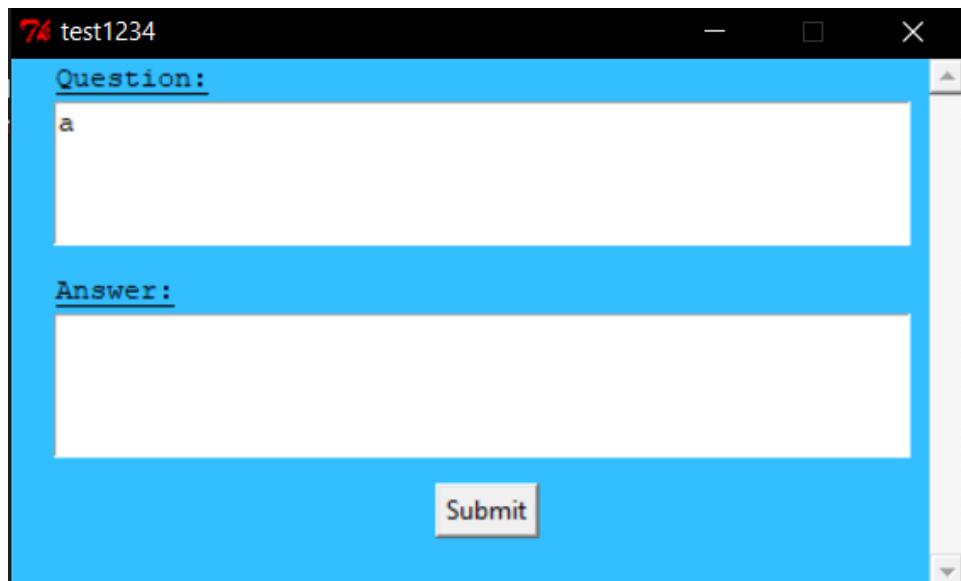
    questionLabel=Label(completeWindow,text="Question:",bg="#33BFFF",
    questionLabel.place(x=20,y=0)

    questionBox=Text(completeWindow,width=50,height=4,wrap=WORD)
    questionBox.place(x=20,y=20)

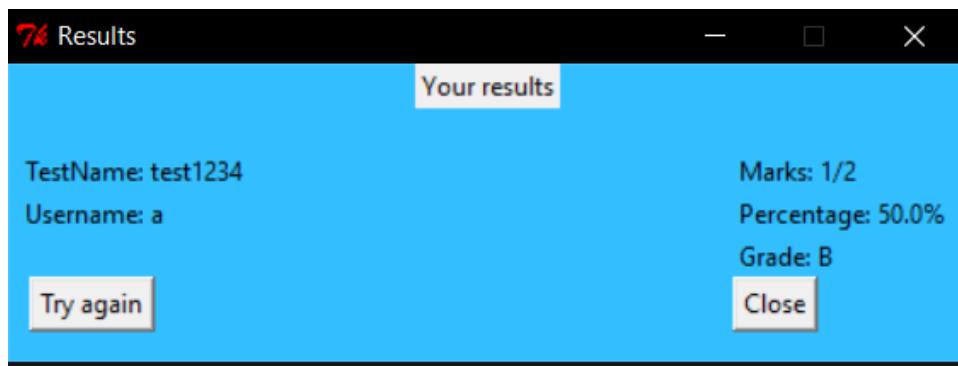
    questionBox.insert(END,questions[0])

    questionBox.config(yscrollcommand=scrollbar.set,state=DISABLED)
    scrollbar.config(command=questionBox.yview)

    answerLabel=Label(completeWindow,text="Answer:",bg="#33BFFF",font
```



```
def ResultsGUI(grade,marks,username,testName,  
    results=Tk()  
    results.geometry("450x140")  
    results.title("Results")  
    results.configure(bg="#33BFFF")  
  
    results.resizable(False,False)  
  
    labelResultsMenu=Label(results,text="Your  
    labelResultsMenu.pack(side=TOP)  
  
    labelTestName=Label(results,bg="#33BFFF",  
    labelTestName.place(x=5,y=40)  
  
    labelUsername=Label(results,bg="#33BFFF",  
    labelUsername.place(x=5,y=60)  
  
    labelMarks=Label(results,bg="#33BFFF",tex  
    labelMarks.place(x=340,y=40)  
  
    percentage=100*float(marks)/float(totalMa  
  
    labelPercent=Label(results,bg="#33BFFF",t  
    labelPercent.place(x=340,y=60)  
  
    labelGrade=Label(results,bg="#33BFFF",tex
```



(after)

```
def testCompletion(testName,questions,answers,rowOfEntry,rows):
    completeWindow=Tk()
    completeWindow.geometry("450x250")
    completeWindow.title(testName)
    completeWindow.configure(bg="#00FF6E")

    completeWindow.resizable(False,False)

    scrollbar=Scrollbar(completeWindow)
    scrollbar.pack(side=RIGHT,fill=Y)

    questionLabel=Label(completeWindow,text="Question:",bg="#00FF6E",
    questionLabel.place(x=20,y=0)

    questionBox=Text(completeWindow,width=50,height=4,wrap=WORD)
    questionBox.place(x=20,y=20)

    questionBox.insert(END,questions[0])

    questionBox.config(yscrollcommand=scrollbar.set,state=DISABLED)
    scrollbar.config(command=questionBox.yview)

    answerLabel=Label(completeWindow,text="Answer:",bg="#00FF6E",font=
    answerLabel.place(x=20,y=100)
```

Hex values changed for a different colour



```

def ResultsGUI(grade,marks,username,testName
    results=Tk()
    results.geometry("450x140")
    results.title("Results")
    results.configure(bg="#00FF6E")Hex values changed for a different colour
    results.resizable(False,False)

    labelResultsMenu=Label(results,text="Your results")
    labelResultsMenu.pack(side=TOP)

    labelTestName=Label(results,bg="#00FF6E")
    labelTestName.place(x=5,y=40)

    labelUsername=Label(results,bg="#00FF6E")
    labelUsername.place(x=5,y=60)

    labelMarks=Label(results,bg="#00FF6E",text="Marks: "+str(marks))
    labelMarks.place(x=340,y=40)

    percentage=100*float(marks)/float(totalMarks)
    labelPercent=Label(results,bg="#00FF6E",text="Percentage: "+str(percentage)+"%")
    labelPercent.place(x=340,y=60)

    labelGrade=Label(results,bg="#00FF6E",text="Grade: "+grade)
    labelGrade.place(x=340,y=80)

```



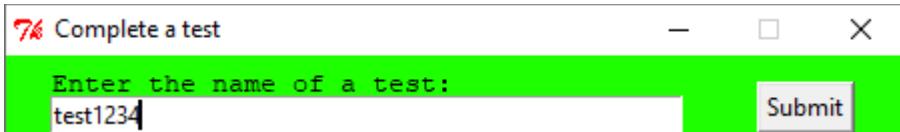
I also modify the nameTestFile function, this function was first made for naming the test file to be created, but also serves to name the test file to complete, because of this I need to change the background colour when the function is used to complete a test so that it follows the same colour scheme.

(before)

```

elif function=="Complete":
    submitButton=Button(fileCreator,text="Submit",command=lambda:checkFileName(nameOfTextBox,username))
    submitButton.place(x=375,y=13)

```

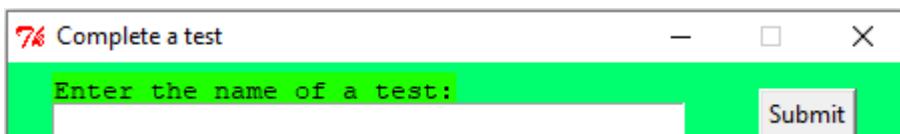


(after)

```
elif function=="Complete":
    submitButton=Button(fileCreator,text="Submit",command=lambda:checkFileName(nameOfTextBox,username))
    submitButton.place(x=375,y=13)

fileCreator.configure(bg="#00FF6E")
```

Coded added to follow the colour scheme for test completion if the function is used with the correct parameters



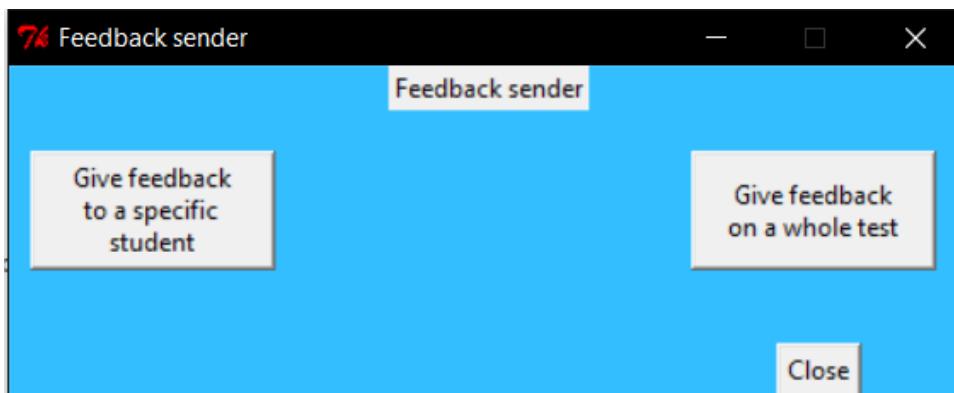
## Feedback sender version 2

First, I will introduce a new colour scheme to the feedback sender GUIs. This was suggested by my client during the post client interview with teachers. **Otherwise, the program would look less aesthetically pleasing.** I will change the colour of each respective system, e.g.: red for registering, yellow for logging in.

(before)

```
def feedbackSelectorGUI():
    username="**"

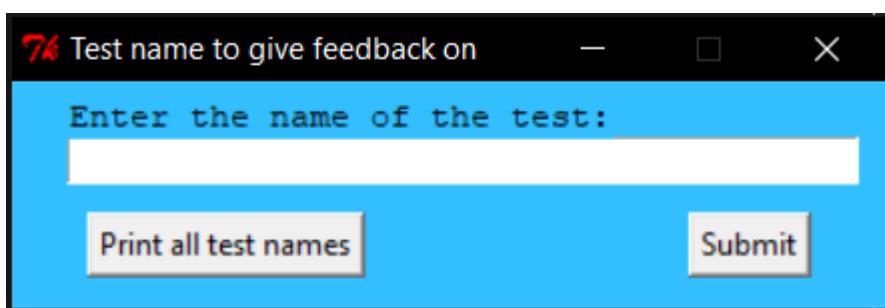
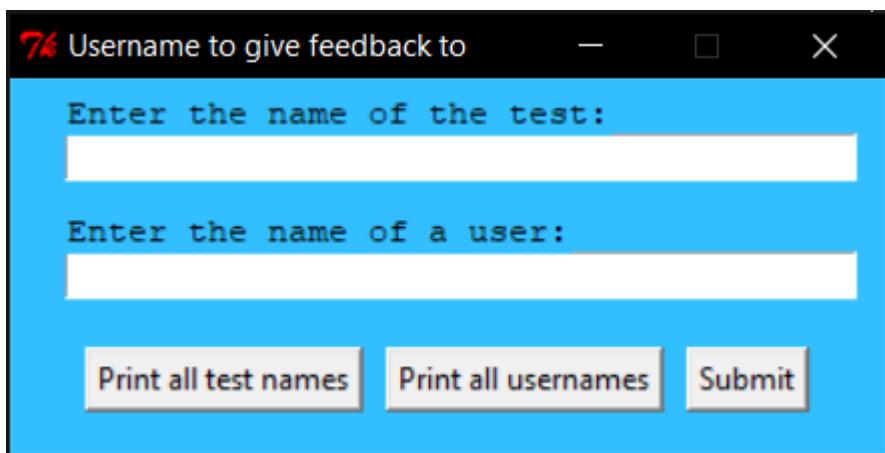
    feedbackSelector=Tk()
    feedbackSelector.geometry("450x160")
    feedbackSelector.title("Feedback sender")
    feedbackSelector.configure(bg="#33BFFF")
```



```
def nameFeedbackReciever(function,usernamesList,testNamesList):
    nameFeedbackReciever=Tk()
    nameFeedbackReciever.geometry("350x90")
    nameFeedbackReciever.configure(bg="#33BFFF")

label1Function=Label(nameFeedbackReciever,text="Enter the name of the test:",bg="#33BFFF",
label1Function.place(x=22,y=5)

label2Function=Label(nameFeedbackReciever,text="Enter the name of a user:",bg="#33BFFF",
label2Function.place(x=22,y=52)
```



```
def printList(function,listToPrint):
    displayList=Tk()
    displayList.geometry("450x200")
    displayList.title("Username list")
    displayList.configure(bg="#33BFFF")

    displayList.resizable(False,False)

    scrollbar=Scrollbar(displayList)
    scrollbar.pack(side=RIGHT,fill=Y)

    listToDisplay=Text(displayList,bg="#33BFFF",
```

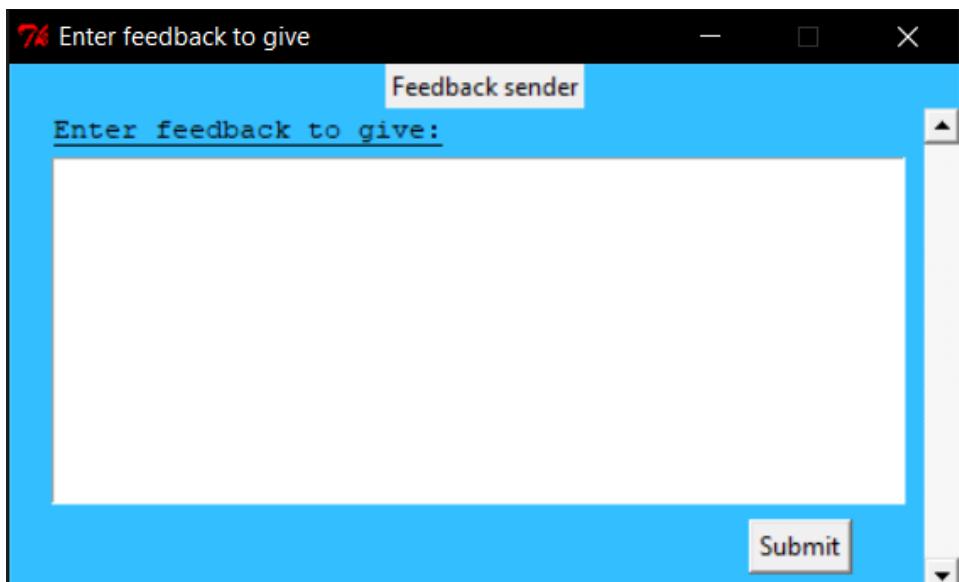


```
def giveFeedbackGUI(testName,username):
    giveFeedbackGUI=Tk()
    giveFeedbackGUI.geometry("450x250")
    giveFeedbackGUI.title("Enter feedback to give")
    giveFeedbackGUI.configure(bg="#33BFFF")

    giveFeedbackGUI.resizable(False,False)

    labelFeedbackSender=Label(giveFeedbackGUI,text="Feedback sender")
    labelFeedbackSender.pack(side=TOP)

    labelFunction=Label(giveFeedbackGUI,text="Enter feedback to give:",bg="#33BFFF",
    ...
```

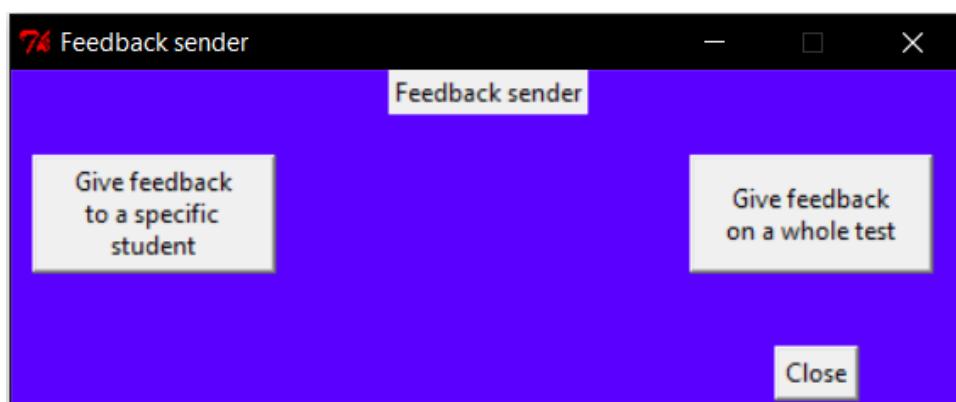


(after)

```
def feedbackSelectorGUI():
    username="**"

    feedbackSelector=Tk()
    feedbackSelector.geometry("450x160")
    feedbackSelector.title("Feedback sender")
    feedbackSelector.configure(bg="#5900FF")
```

Hex value changed  
for a different  
colour

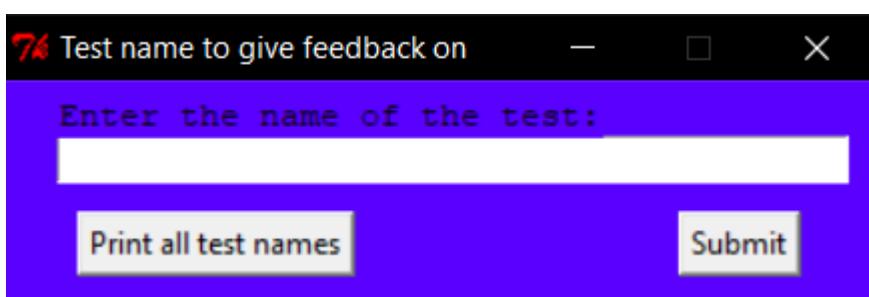
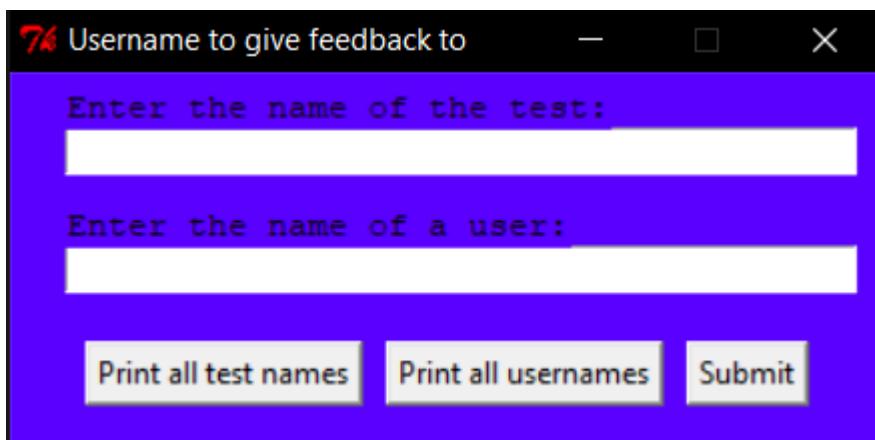


```
def nameFeedbackReciever(function,usernamesList,testNamesList):
    nameFeedbackReciever=Tk()
    nameFeedbackReciever.geometry("350x90")
    nameFeedbackReciever.configure(bg="#5900FF")
```

Hex value  
changed for a  
different colour

```
labelFunction=Label(nameFeedbackReciever,text="Enter the name of the test:",bg="#5900FF",
labelFunction.place(x=22,y=5)
```

```
label2Function=Label(nameFeedbackReciever,text="Enter the name of a user:",bg="#5900FF")
label2Function.place(x=22,y=52)
```



```
def printList(function,listToPrint):
    displayList=Tk()
    displayList.geometry("450x200")
    displayList.title("Username list")
    displayList.configure(bg="#5900FF")

    displayList.resizable(False,False)

    scrollbar=Scrollbar(displayList)
    scrollbar.pack(side=RIGHT,fill=Y)

    listToDisplay=Text(displayList,bg="#5900FF",
```

Hex values changed to  
give a different colour



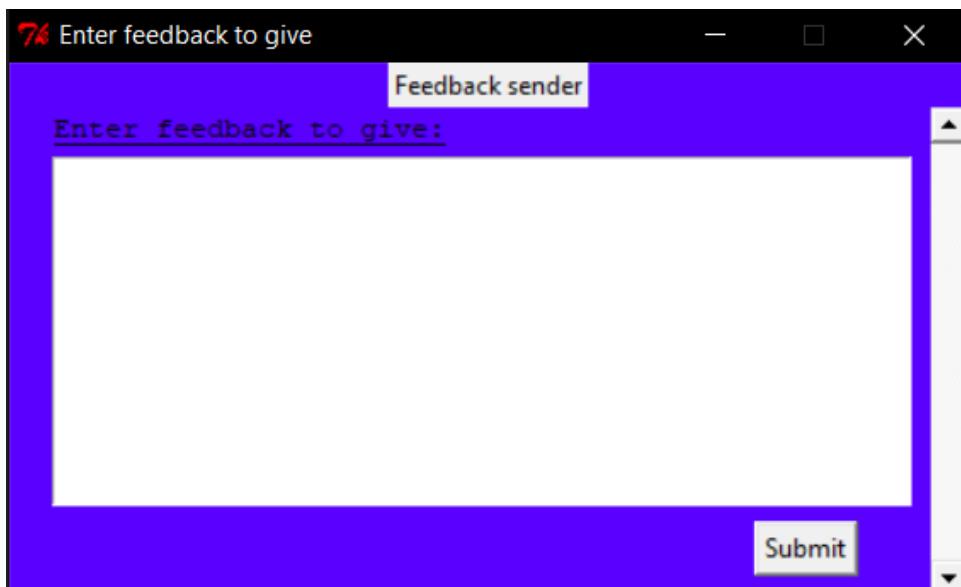
```
def giveFeedbackGUI(testName,username):
    giveFeedbackGUI=Tk()
    giveFeedbackGUI.geometry("450x250")
    giveFeedbackGUI.title("Enter feedback to give")
    giveFeedbackGUI.configure(bg="#5900FF")

    giveFeedbackGUI.resizable(False,False)

    labelFeedbackSender=Label(giveFeedbackGUI,text="Feedback sender")
    labelFeedbackSender.pack(side=TOP)

    labelFunction=Label(giveFeedbackGUI,text="Enter feedback to give:",bg="#5900FF")
```

Hex values changed to  
give a different colour



## Feedback receiver version 2

First, I will introduce a new colour scheme to the feedback receiver GUI. This was suggested by my client during the post client interview with teachers. **Otherwise, the program would look less aesthetically pleasing.** I will change the colour of each respective system, e.g.: red for registering, yellow for logging in.

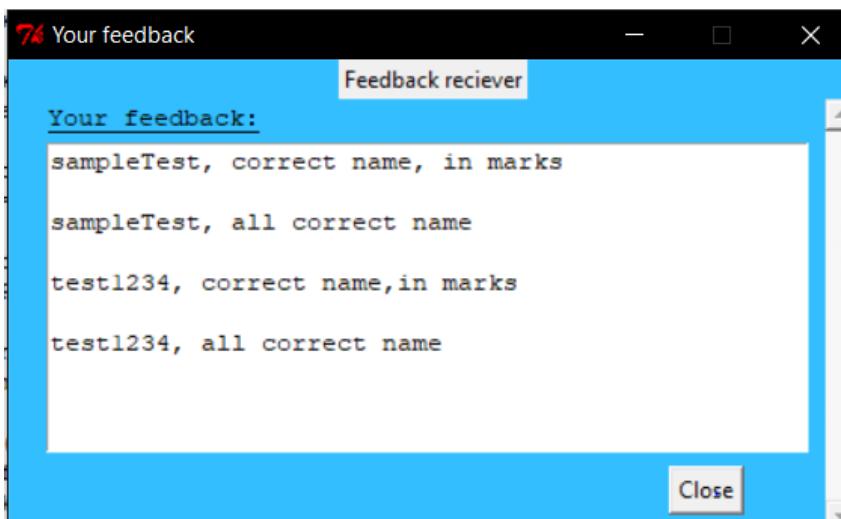
(before)

```
def displayFeedback(feedback):
    displayFeedback=Tk()
    displayFeedback.geometry("450x250")
    displayFeedback.title("Your feedback")
    displayFeedback.configure(bg="#33BFFF")

    displayFeedback.resizable(False, False)

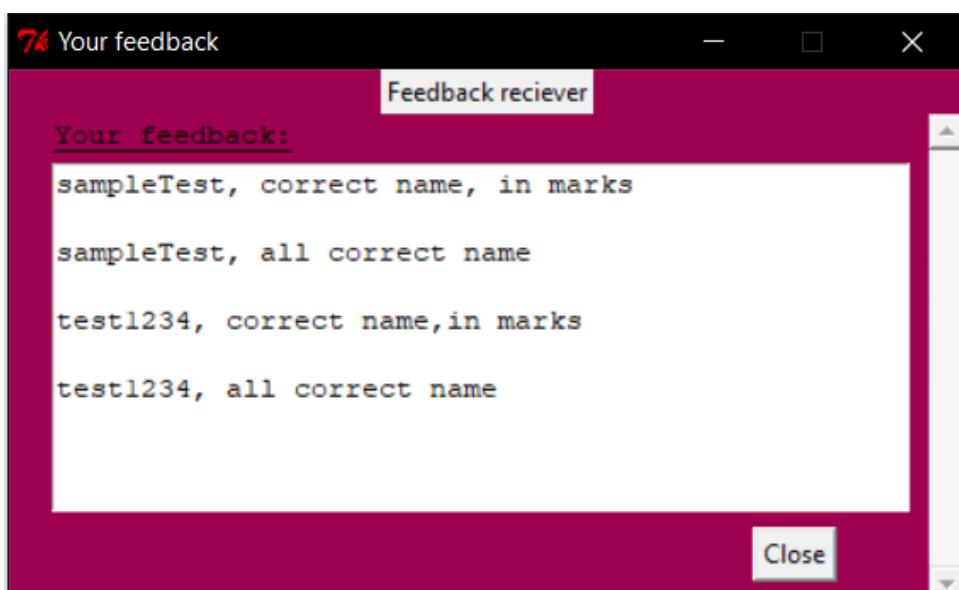
    labelFeedbackReciever=Label(displayFeedback, text="Feedback receiver")
    labelFeedbackReciever.pack(side=TOP)

    labelFunction=Label(displayFeedback, text="Your feedback:", bg="#33BFFF",
```



(after)

```
def displayFeedback(feedback):  
    displayFeedback=Tk()  
    displayFeedback.geometry("450x250")  
    displayFeedback.title("Your feedback")  
    displayFeedback.configure(bg="#9E004F")  
  
    displayFeedback.resizable(False, False)  
  
    labelFeedbackReciever=Label(displayFeedback, text="Feedback reciever")  
    labelFeedbackReciever.pack(side=TOP)  
  
    labelFunction=Label(displayFeedback, text="Your feedback:", bg="#9E004F",
```



Next, I will add a message box that appears if the user has no feedback. **Otherwise, the user can't tell if they have any feedback**

(getFeedback function)

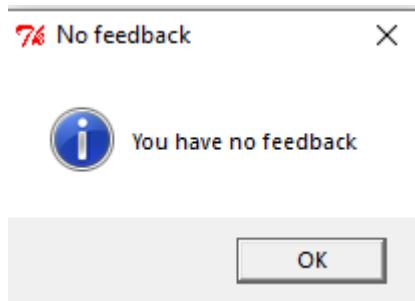
(before)

```
# Opens the feedback GUI for the user to view the feedback
displayFeedback(feedback)
```

(after)

```
# Opens the feedback GUI for the user to view the feedback
if len(feedback) !=0:
    displayFeedback(feedback)

else:
    messagebox.showinfo("No feedback","You have no feedback")
```



## Final program code

```
# All additional modules
from tkinter import *
import csv
import os
import shutil

#
# MAIN MENU
#
# The main elements of the main menu are defined

root=Tk()
root.geometry("450x200")
root.title("Main Menu")
root.configure(bg="#33BFFF")
```

```
root.resizable(False,False)

labelMenu=Label(root,text="Welcome to the main menu")
labelMenu.pack(side=TOP)

#
# FEEDBACK RECIEVER
#
# Reads all relevant feedback to an array to be added to the window, so that the user can see it
def getFeedback(username,listOfFiles):
    relevantFiles=[]
    feedback=[]

    # Searches all files that end with " - marks.csv" and looks for if there is an entry with the logged in user's username
    for i in range(0,len(listOfFiles)):
        currentFile=listOfFiles[i]+" - marks.csv"
        csvFile=open(currentFile,'r+')
        csvReader=csv.reader(csvFile)
        rows=list(csvReader)
        csvFile.close()

        # Adds all relevant files to an array
        for j in range(1,len(rows)):
            if username==rows[j][0]:
                relevantFiles.append(listOfFiles[i])
                break

    # Searches array of all relevant files for any feedback either directed to the logged in user, or directed to all users who have completed the test
    for i in range(0,len(relevantFiles)):
        currentFile=relevantFiles[i]+" - feedback.csv"
        csvFile=open(currentFile,'r+')
        csvReader=csv.reader(csvFile)
        rows=list(csvReader)
```

```
csvFile.close()

# Adds all feedback that is relevant to the user to an array

for j in range(1,len(rows)):

    if username==rows[j][0] or rows[j][0]=="*":

        feedback.append(rows[j][1])



print(relevantFiles)

print(feedback)

# Opens the feedback GUI for the user to view the feedback

displayFeedback(feedback)

# Displays feedback in a GUI

def displayFeedback(feedback):

    displayFeedback=Tk()

    displayFeedback.geometry("450x250")

    displayFeedback.title("Your feedback")

    displayFeedback.configure(bg="#9E004F")



    displayFeedback.resizable(False,False)




labelFeedbackReciever=Label(displayFeedback,text="Feedback receiver")

labelFeedbackReciever.pack(side=TOP)

labelFunction=Label(displayFeedback,text="Your feedback:",bg="#9E004F",font=("Courier 10 underline"), borderwidth=0)

labelFunction.place(x=20,y=22)

# Text widget is used to insert feedback into the GUI to display it to the user

displayedFeedback=Text(displayFeedback,width=50,height=10,wrap=WORD)

displayedFeedback.place(x=20,y=44)

# Button to close the GUI

submitButton=Button(displayFeedback,text="Close",command=lambda: displayFeedback.destroy())

submitButton.place(x=350,y=215)
```

```
scrollbar=Scrollbar(displayFeedback)
scrollbar.pack(side=RIGHT,fill=Y)

displayedFeedback.config(yscrollcommand=scrollbar.set)
scrollbar.config(command=displayedFeedback.yview)

for i in range(0,len(feedback)):
    currentFeedback=feedback[i]+"\n\n"
    displayedFeedback.insert(END,currentFeedback)
#
# FEEDBACK GIVER
#
# Allows user to select target to give feedback to
def feedbackSelectorGUI():
    username="*"

    feedbackSelector=Tk()
    feedbackSelector.geometry("450x160")
    feedbackSelector.title("Feedback sender")
    feedbackSelector.configure(bg="#5900FF")

    feedbackSelector.resizable(False,False)

    labelFeedbackMenu=Label(feedbackSelector,text="Feedback sender")
    labelFeedbackMenu.pack(side=TOP)

    # Button to start the process to give specific student feedback
    specificButton=Button(feedbackSelector,width=15,height=3,wraplength=80,text="Give feedback to a specific student",command=lambda: getUsersAndTestNames("Specific",username))
    specificButton.place(x=10,y=40)

    # Button to start the process to give whole test feedback
    wholeButton=Button(feedbackSelector,width=15,height=3,wraplength=80,text="Give feedback on a whole test",command=lambda: getUsersAndTestNames("Test",username))
    wholeButton.place(x=320,y=40)
```

```
# Button to close the GUI
closeButton=Button(feedbackSelector,text="Close",command=lambda: feedbackSelector.destroy())
closeButton.place(x=360,y=130)

# Gets usernames and test names for button functions in the next GUI
def getUsernamesAndTestNames(function,username):
    # Creates a list of all files in the current directory
    rootFile=os.path.abspath(os.curdir)
    listOffiles=os.listdir(rootFile)
    print(listOffiles)

    testNamesList=[]
    usernamesList=[]

    # Searches all files on the current directory for files ending in "grades.csv", as this indicates their is a fully created test
    # List of found test files are added to the array testNamesList
    for i in range(0,len(listOffiles)):
        file=listOffiles[i]
        if file[-10:]==".grades.csv":
            testNamesList.append(file[0:-13])

    print(testNamesList)

    # Starts the process to recieve feedback
    if function=="Recieve":
        getFeedback(username,testNamesList)

    # Starts the process to give feedback on an entire test
    if function=="Test":
        nameFeedbackReciever("Test",usernamesList,testNamesList)

    # Starts the process to give feedback to a specific student
    if function=="Specific":
        csvFile=open('login.csv','r+')
        csvReader=csv.reader(csvFile)
        rows=list(csvReader)
```

```
print(rows)

# List of all usernames is created using the 'login.csv' file

for i in range(1,len(rows)):

    username=rows[i][0]

    usernamesList.append(username)

print(usernamesList)

csvFile.close()

nameFeedbackReciever("Specific",usernamesList,testNamesList)

# Displays either a list of usernames or a list of test names using a window

def printList(function,listToPrint):

    displayList=Tk()

    displayList.geometry("450x200")

    displayList.title("Username list")

    displayList.configure(bg="#5900FF")

    displayList.resizable(False,False)

    scrollbar=Scrollbar(displayList)

    scrollbar.pack(side=RIGHT,fill=Y)

    # Text widget to insert the list that is to be displayed

    listToDisplay=Text(displayList,bg="#5900FF", width=450,wrap=WORD)

    listToDisplay.pack()

    listToDisplay.insert(END,listToPrint)

    listToDisplay.config(yscrollcommand=scrollbar.set,state=DISABLED)

    scrollbar.config(command=listToDisplay.yview)

# Title is changed depending on the function, as this function is used when giving specific and whole feedback

if function=="Specific":
```

```
displayList.title("Username list")

elif function=="Test":
    displayList.title("Test names list")

# Checks if inputted test name exists
def checkFeedbackTest(nameOfFeedbackTest,testNamesList):
    username="*"
    testName=nameOfFeedbackTest.get()

    # Checks the inputted test name exists and displays a relevant error message if it does not exist or the input is invalid
    if testName=="":
        messagebox.showerror("Invalid input","Please enter the test name into the input box")
    else:
        for i in range(0,len(testNamesList)):
            if testName==testNamesList[i]:
                giveFeedbackGUI(testName,username)
                break

    elif testName!=testNamesList[i] and i==len(testNamesList)-1:
        messagebox.showerror("Test has not been found","The entered test has not been found, please try again")

#Checks if inputted test name and username exists
def checkFeedbackSpecific(nameOfFeedbackTest,nameOfFeedbackUser,testNamesList,usernamesList):
    actualUsername=nameOfFeedbackUser.get()
    actualTestName=nameOfFeedbackTest.get()
    username=""
    testName=""

    # Checks if both the entered username and test name exist and are not invalid inputs. Provides relevant error messages.
    if actualUsername=="" or actualTestName=="":
        messagebox.showerror("Invalid input","Please make sure both input boxes are filled")

    else:
        for i in range(0,len(testNamesList)):
            if actualTestName==testNamesList[i]:
```

```
testName=actualTestName
break

elif actualTestName!=testNamesList[i] and i==len(testNamesList)-1:
    messagebox.showerror("Test has not been found","The entered test has not been found, please try again")

for j in range(0,len(usernamesList)):
    if actualUsername==usernamesList[j]:
        username=actualUsername
        break

    elif actualUsername!=usernamesList[j] and j==len(usernamesList)-1:
        messagebox.showerror("Username has not been found","The entered username has not been found, please try again")

if username==actualUsername and testName==actualTestName:
    giveFeedbackGUI(testName,username)

# Saves inputted feedback to the appropriate .csv file
def saveFeedback(testName,username,feedback):
    testName=testName+" - feedback.csv"
    feedback=feedback.get(1.0,'end-1c')

    if feedback=="":
        messagebox.showerror("Invalid input","Please enter feedback into the input box")

    # Opens feedback file linked to the test, copies all the contents, adds to the contents, then re-adds the contents to the .csv file
    elif feedback!="":
        csvFile=open(testName)
        csvReader=csv.reader(csvFile)
        rows=list(csvReader)

        rows.append([username,feedback])

        csvFile.close()
        csvFile=open(testName,'w',newline="")
```

```
csvWriter=csv.writer(csvFile)
csvWriter.writerows(rows)

csvFile.close()

messagebox.showinfo("Feedback saved","Feedback has been added successfully")

# Creates and opens the GUI to give feedback
def giveFeedbackGUI(testName,username):
    giveFeedbackGUI=Tk()
    giveFeedbackGUI.geometry("450x250")
    giveFeedbackGUI.title("Enter feedback to give")
    giveFeedbackGUI.configure(bg="#5900FF")

    giveFeedbackGUI.resizable(False,False)

    labelFeedbackSender=Label(giveFeedbackGUI,text="Feedback sender")
    labelFeedbackSender.pack(side=TOP)

    labelFunction=Label(giveFeedbackGUI,text="Enter feedback to give:",bg="#5900FF",font=("Courier 10 underline"), borderwidth=0)
    labelFunction.place(x=20,y=22)

    # Text widget to allow the user to enter feedback
    feedback=Text(giveFeedbackGUI,width=50,height=10,wrap=WORD)
    feedback.place(x=20,y=44)

    # Button to start the process to save the entered feedback
    submitButton=Button(giveFeedbackGUI,text="Submit",command=lambda: saveFeedback(testName,username,feedback))
    submitButton.place(x=350,y=215)

    scrollbar=Scrollbar(giveFeedbackGUI)
    scrollbar.pack(side=RIGHT,fill=Y)

    feedback.config(yscrollcommand=scrollbar.set)
    scrollbar.config(command=feedback.yview)
```

```
# Allows user to enter who the feedback is for

def nameFeedbackReciever(function,usernamesList,testNamesList):

    nameFeedbackReciever=Tk()
    nameFeedbackReciever.geometry("350x90")
    nameFeedbackReciever.configure(bg="#5900FF")

    nameFeedbackReciever.resizable(False,False)

# Entry to allow the user to enter the name of the test to give feedback on

nameOfFeedbackTest=Entry(nameFeedbackReciever,width=52)
nameOfFeedbackTest.place(x=22,y=22)

label1Function=Label(nameFeedbackReciever,text="Enter the name of the test:",bg="#5900FF",font=("Courier 10"), borderwidth=0)
label1Function.place(x=22,y=5)

# Button to display all compatible tests in the current directory

print1Button=Button(nameFeedbackReciever,text="Print all test names",command=lambda: printList("Test",testNamesList))
print1Button.place(x=30,y=52)

# If feedback is being given for an entire test, less elements of the GUI are defined as they are unnecessary

if function=="Test":

    # Submit button to start the check the inputs when giving whole test feedback

    submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda: checkFeedbackTest(nameOfFeedbackTest,testNamesList))
    submitButton.place(x=270,y=52)

    nameFeedbackReciever.title("Test name to give feedback on")

# If feedback is being given to a specific student, more elements of the GUI are defined as they are necessary to send the feedback to the specific user

elif function=="Specific":

    nameFeedbackReciever.title("Username to give feedback to")
    nameFeedbackReciever.geometry("350x150")

    # Entry to allow the user to enter the username of the student to give feedback to

    nameOfFeedbackUser=Entry(nameFeedbackReciever,width=52)
    nameOfFeedbackUser.place(x=22,y=69)

    label2Function=Label(nameFeedbackReciever,text="Enter the name of a user:",bg="#5900FF",font=("Courier 10"), borderwidth=0)
```

```
label2Function.place(x=22,y=52)

# Submit button to start the check the inputs when giving specific student feedback
submitButton=Button(nameFeedbackReciever,text="Submit",command=lambda:
checkFeedbackSpecific(nameOfFeedbackTest,nameOfFeedbackUser,testNamesList,usernamesList))

submitButton.place(x=270,y=107)

print1Button.place(x=30,y=107)

# Button to display all usernames registered in the 'login.csv' file
print2Button=Button(nameFeedbackReciever,text="Print all usernames",command=lambda: printList("Specific",usernamesList))

print2Button.place(x=150,y=107)

#
# TEST COMPLETING
#
# Reads the test csv file (also used to delete the last row of a test during the test creation process)
def readTestFile(testName,mode,username):
    # Contents of the test .csv file are read
    testFile=testName+".csv"
    csvFile=open(testFile,'r+')
    csvFileContent=(csvFile)
    next(csvFile)

    csvList=[]

    # Contents of the .csv file are turned into a dictionary format
    for row in csvFileContent:
        csv=row.strip().split(",")
        heading=['Question','Answer']
        data=zip(heading,csv)
        DataDict=dict(data)
        csvList.append(DataDict)

# Deletes last row of the test (test creation)
```

```
if mode=="Delete":  
    if len(csvList)!=0:  
        csvList.pop()  
        saveTestData(csvList,testFile,DataDict,'w+')  
    else:  
        messagebox.showerror("File is empty","There are no more entries to delete")  
  
# Reads all the questions and answers  
  
elif mode=="Read":  
    readAllQuestions(testName,testFile,username)  
  
  
# Checks that the test file exists  
  
def checkFileName(testName,username):  
    testName=testName.get()  
  
    # Finds the current directory and adds the test name to the file path  
    rootFile=os.path.abspath(os.curdir)  
  
    file=rootFile+"\\"+testName+".csv"  
  
    # Checks if there is a path to the file to see if it exists  
  
    if os.path.isfile(file)==True:  
        readTestFile(testName,"Read",username)  
    elif os.path.isfile(file)==False:  
        messagebox.showerror("File not found","The file has not been found, please try again")  
  
# Adds new user test attempt to the marks file  
  
def setMarkFile(username,testName,questions,answers):  
    file=testName+" - marks.csv"  
  
    csvFile=open(file,'a')  
  
    newLine=username+",0,U\n"  
  
    csvFile.write(newLine)  
  
    csvFile.close()  
  
  
    file=testName+" - marks.csv"  
    csvFile=open(file,'r+')  
  
    csvReader=csv.reader(csvFile)
```

```
rows=list(csvReader)

# Finds the last recorded entry by the user, so that only the most recent test attempt by the user is modified
lastAttemptByUser=1

print(rows)

for i in range(len(rows)-1,1,-1):
    if rows[i][0]==username:
        lastAttemptByUser=i
        break

print(lastAttemptByUser)
csvFile.close()

testCompletion(testName,questions,answers,lastAttemptByUser,rows)

# Reads all questions and answers in the test file
def readAllQuestions(testName,testFile,username):
    questions=[]
    answers=[]
    temp=[]

    # Opens the test .csv file and reads the contents
    csvFile=open(testFile)
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)

    # Adds all the questions into a list
    i=1
    while i!=len(rows):
        question=rows[i][0]
        questions.append(question)
        i=i+1

    print(questions)
```

```
# Adds all the answers to a list of lists, because there can be multiple answers linked to one question
```

```
i=1  
j=1  
while i!=len(rows):  
    while j!=len(rows[i]):  
        answer=rows[i][j]  
        temp.append(answer)  
        j=j+1  
    answers.append(temp)  
    temp=[]  
    j=1  
    i=i+1  
  
print(answers)
```

```
# Adds the correct headings to the mark file, if they have not been added already
```

```
openMarkOrFeedbackFile(testName+' - marks.csv','Mark')
```

```
# Adds a new test attempt into the mark file
```

```
setMarkFile(username,testName,questions,answers)
```

```
# Checks the inputted answer against the actual answers
```

```
def checkAnswer(answer,answers,i,testName,rowOfEntry,rows):
```

```
file=testName+" - marks.csv"  
csvFile=open(file,'w',newline='')  
csvWriter=csv.writer(csvFile)
```

```
correctAnswers=answers[i]
```

```
print(rowOfEntry)
```

```
j=0
```

```
# Checks if the input answer matches the correct answer or answers
```

```
while j!=len(correctAnswers):
    # If the answer is correct the user's most recent entry in the mark file will gain an additional mark
    if answer==correctAnswers[j]:
        rows[rowOfEntry][1]=int(rows[rowOfEntry][1])+1
        csvWriter.writerows(rows)
        print("correct")
        break

    # If the answer is incorrect, then the function re-adds the contents of the mark file
    elif answer!=correctAnswers[j] and j+1==len(correctAnswers):
        csvWriter.writerows(rows)
        print("wrong")
        j=j+1

csvFile.close()

# Displays GUI that shows results of test
def ResultsGUI(grade,marks,username,testName,totalMarks):
    results=Tk()
    results.geometry("450x140")
    results.title("Results")
    results.configure(bg="#00FF6E")

    results.resizable(False,False)

    labelResultsMenu=Label(results,text="Your results")
    labelResultsMenu.pack(side=TOP)

    labelTestName=Label(results,bg="#00FF6E",text="TestName: "+testName)
    labelTestName.place(x=5,y=40)

    labelUsername=Label(results,bg="#00FF6E",text="Username: "+username)
    labelUsername.place(x=5,y=60)

    labelMarks=Label(results,bg="#00FF6E",text="Marks: "+str(marks)+"/"+str(totalMarks))
    labelMarks.place(x=340,y=40)
```

```
percentage=100*float(marks)/float(totalMarks)

labelPercent=Label(results,bg="#00FF6E",text="Percentage: "+str(percentage)+"%")
labelPercent.place(x=340,y=60)

labelGrade=Label(results,bg="#00FF6E",text="Grade: "+grade)
labelGrade.place(x=340,y=80)

# Button to close the GUI
closeButton=Button(results,text="Close",command=lambda: results.destroy())
closeButton.place(x=340,y=100)

# Button to start the test completion process again with the same test
tryAgainButton=Button(results,text="Try again",command=lambda: readTestFile(testName,"Read",username))
tryAgainButton.place(x=10,y=100)

# Adds new grade to marks file and prepares results screen
def updateGrade(testName,rowOfEntry):
    markFile=testName+" - marks.csv"
    csvFile=open(markFile,'r+')
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)
    csvFile.close()

    print(rows)

    # Finds the marks of the user's most recent test attempt
    marks=rows[rowOfEntry][1]
    print(marks)

    gradeFile=testName+" - grades.csv"
    csvFile=open(gradeFile,'r+')
    csvReader=csv.reader(csvFile)
    grades=list(csvReader)
```

```
print(grades)
```

```
grade=""
```

```
i=0
```

```
# Finds the grade based on the user's marks
```

```
for i in range(0,5):
```

```
    if marks>=grades[1][i]:
```

```
        grade=grades[0][i]
```

```
        break
```

```
    else:
```

```
        i=i+1
```

```
print(grade)
```

```
csvFile.close()
```

```
# Saves the new grade
```

```
rows[rowOfEntry][2]=grade
```

```
username=rows[rowOfEntry][0]
```

```
csvFile=open(markFile,'w',newline="")
```

```
csvWriter=csv.writer(csvFile)
```

```
csvWriter.writerows(rows)
```

```
csvFile.close()
```

```
totalMarks=0
```

```
testFile=testName+".csv"
```

```
contents=open(testFile)
```

```
fileContents=(contents)
```

```
next(contents)
```

```
print(fileContents)
```

```
for row in fileContents:
    totalMarks=totalMarks+1

print(totalMarks)

ResultsGUI(grade,marks,username,testName,totalMarks)

# Updates the question box after an answer has been submitted
def updateQuestionBox(questionBox,questions,answerBox,answers,testName,rowOfEntry,rows):
    answer=answerBox.get(1.0,'end-1c')
    questionBox.config(state=NORMAL)

    currentQuestion=questionBox.get(1.0,'end-1c')

    questionBox.delete(1.0,END)

    i=0

    # Checks if the current question in the text widget matches the current question in the questions array. Updates the text box with the new question.
    while i!=len(questions):
        if currentQuestion!=questions[i]:
            i=i+1
        elif currentQuestion==questions[i]:
            # Once the next question has been found the inputted answer is checked against the actual answer/answers
            checkAnswer(answer,answers,i,testName,rowOfEntry,rows)
            if i+1<len(questions):
                newQuestion=questions[i+1]
                questionBox.insert(END,newQuestion)
                break
            break

    # If there are no more questions, the process to update the user's marks and grades, and display their results starts
    if i+1==len(questions):
        updateGrade(testName,rowOfEntry)
```



```
answerLabel.place(x=20,y=100)

# Text widget used to allow the user to input their answer
answerBox=Text(completeWindow,width=50,height=4,wrap=WORD)
answerBox.place(x=20,y=120)

# Submit button to check if the inputted answer is correct and display the new question
submitButton=Button(completeWindow,text="Submit",command=lambda:
updateQuestionBox(questionBox,questions,answerBox,answers,testName,rowOfEntry,rows))

submitButton.place(x=200,y=200)

#
# TEST CREATOR
#
# Saves inputted grades to csv file
def saveGrades(grades,file,gradesData,gradeWindow):
    try:
        fileHandle=open(file,'r+')
        fileContent=fileHandle.read()
        if fileContent.strip()=='':
            fileHandle.write('A,B,C,D,E,F\n')
        for item in grades:
            fileHandle.write('{A},{B},{C},{D},{E},{F}'.format(**gradesData))
            fileHandle.write('\n')
        messagebox.showinfo("Success","Grades saved successfully")
        gradeWindow.destroy()
        fileHandle.close()
    except OSError:
        messagebox.showerror("Not saved","Grades not saved successfully, please try again")

# Gets inputted grades from GUI
def getGrades(A,B,C,D,E,F,file,gradeWindow):
    # Gets all the user's grades inputs and adds them to a dictionary
```

```
grades=[]

a=A.get(1.0,'end-1c')
b=B.get(1.0,'end-1c')
c=C.get(1.0,'end-1c')
d=D.get(1.0,'end-1c')
e=E.get(1.0,'end-1c')
f=F.get(1.0,'end-1c')

gradesData={'A':a,'B':b,'C':c,'D':d,'E':e,'F':f}

grades.append(gradesData)

# Checks if the user has entered inputs into all input boxes
if len(a)!=0 and len(b)!=0 and len(c)!=0 and len(d)!=0 and len(e)!=0 and len(f)!=0:
    saveGrades(grades,file,gradesData,gradeWindow)
else:
    messagebox.showerror("Grades not added","Please fill all input boxes and try again")

# Opens the grade file and formats the headings
def openGradeFile(file,A,B,C,D,E,F,gradeWindow):
    csvFile=open(file,'r+')
    csvFile.write('A,B,C,D,E,F\n')
    fileRead=csvFile.read()

    for item in fileRead:
        csvFile.write('{A},{B},{C},{D},{E},{F}'.format(**item))
        csvFile.write('\n')
    csvFile.close()

    getGrades(A,B,C,D,E,F,file,gradeWindow)

# Opens the mark file and formats the headings
def openMarkOrFeedbackFile(fileName,fileType):
    csvFile=open(fileName,'r+')
    fileRead=csvFile.read()

    # Adds the correct headings to the mark file, if they have not been added already
    if fileRead.strip()=='' and fileType=="Mark":
        fileRead=csvFile.read()
```

```
csvFile=open(fileName,'w+')

csvFile.write('Username,Marks,Grades\n')

print("headings have been written")

# Adds the correct headings to the feedback file, if they have not been added already

elif fileRead.strip()=='' and fileType=="Feedback":

    fileRead=csvFile.read()

    csvFile=open(fileName,'w+')

    csvFile.write('Username,Feedback\n')

print("headings have been written")

else:

    print("headings are already written")

# Creates the grade, mark and feedback file by copying and renaming the "empty.csv" file

def gradeFileCreator(testName,A,B,C,D,E,F,gradeWindow):

    rootFile=os.path.abspath(os.curdir)

    gradeFileName=testName.strip(".csv")

    dst1File=gradeFileName+" - grades.csv"

    dst2File=gradeFileName+" - marks.csv"

    dst3File=gradeFileName+" - feedback.csv"

    rootFile=rootFile+"\empty.csv"

    shutil.copy2(rootFile,dst1File)

    shutil.copy2(rootFile,dst2File)

    shutil.copy2(rootFile,dst3File)

    openGradeFile(dst1File,A,B,C,D,E,F,gradeWindow)

    openMarkOrFeedbackFile(dst2File,"Mark")

    openMarkOrFeedbackFile(dst3File,"Feedback")

# Outputs the total marks for a test when in adding grades GUI

def checkMarks(file):

    marks=0
```

```
contents=open(file)
fileContents=(contents)
next(contents)

for row in fileContents:
    marks=marks+1

if marks!=0:
    messagebox.showinfo("Total marks",marks)
else:
    messagebox.showerror("Error","The test is empty please re-create the test")
openCreatedTest(file)

contents.close()

# Grade boundaries GUI
def endTest(closeWindow,file):
    gradeWindow=Tk()
    gradeWindow.geometry("300x150")
    gradeWindow.title("Add grade boundaries")
    gradeWindow.configure(bg="#1EFF00")

    gradeTitle=Label(gradeWindow,text="Add grade boundaries to the test")
    gradeTitle.pack(side=TOP)

    # Text widgets to allow the user to enter grade boundaries
    entryAgrade=Text(gradeWindow,width=5,height=1)
    entryAgrade.place(x=20,y=45)

    entryBgrade=Text(gradeWindow,width=5,height=1)
    entryBgrade.place(x=125,y=45)

    entryCgrade=Text(gradeWindow,width=5,height=1)
    entryCgrade.place(x=230,y=45)
```

```
entryDgrade=Text(gradeWindow,width=5,height=1)
entryDgrade.place(x=20,y=95)

entryEgrade=Text(gradeWindow,width=5,height=1)
entryEgrade.place(x=125,y=95)

entryFgrade=Text(gradeWindow,width=5,height=1)
entryFgrade.place(x=230,y=95)

gradeAlabel=Label(gradeWindow,bg="#1EFF00",text="A:")
gradeAlabel.place(x=20,y=25)

gradeBlabel=Label(gradeWindow,bg="#1EFF00",text="B:")
gradeBlabel.place(x=125,y=25)

gradeClabel=Label(gradeWindow,bg="#1EFF00",text="C:")
gradeClabel.place(x=230,y=25)

gradeDlabel=Label(gradeWindow,bg="#1EFF00",text="D:")
gradeDlabel.place(x=20,y=75)

gradeElabel=Label(gradeWindow,bg="#1EFF00",text="E:")
gradeElabel.place(x=125,y=75)

gradeFlabel=Label(gradeWindow,bg="#1EFF00",text="F:")
gradeFlabel.place(x=230,y=75)

# Button to display a message box with the total number of marks for the test
checkButton=Button(gradeWindow,text="Check total marks",command=lambda: checkMarks(file))
checkButton.place(x=20,y=123)

# Button to start the process to save the grades to a .csv file
submitButton=Button(gradeWindow,text="Submit",command=lambda:
gradeFileCreator(file,entryAgrade,entryBgrade,entryCgrade,entryDgrade,entryEgrade,entryFgrade,gradeWindow))
submitButton.place(x=230,y=123)

gradeWindow.resizable(False,False)
```

```
closeWindow.destroy()

# Displays window representation of the test csv file

def testPreview(file):
    previewWindow=Tk()
    previewWindow.geometry("450x250")
    previewWindow.title("Test preview")
    previewWindow.configure(bg="#1EFF00")

    test=open(file,'r+')
    testData=test.read()

    scrollbar=Scrollbar(previewWindow)
    scrollbar.pack(side=RIGHT,fill=Y)

    # Text widget to insert the contents of the .csv file, to display the test to the user
    testPreview=Text(previewWindow,bg="#1EFF00",width=450,wrap=WORD)
    testPreview.pack()

    testPreview.insert(END,testData)

    testPreview.config(yscrollcommand=scrollbar.set,state=DISABLED)
    scrollbar.config(command=testPreview.yview)

    test.close()

# Saves inputted questions and answers to the csv file

def saveTestData(csv,file,csvData,mode):
    try:
        fileHandle=open(file,mode)
        fileContent=fileHandle.read()

        # If the file is empty, the necessary headings are added
        if fileContent.strip()=='':
            fileHandle.write('Question,Answer\n')

        # Adds the questions and answers from the dictionary
        for item in csv:
```

```
fileHandle.write('{Question},{Answer}'.format(**csvData))

fileHandle.write('\n')

# Displays relevant message box on success based on the arguments used in the function call

if mode=='r+':

    messagebox.showinfo("Success","Question and answers added successfully")

elif mode=='w+':

    messagebox.showinfo("Success","The previous entries have been removed successfully")

fileHandle.close()

except OSError:

    print('Can\'t write to file')

# Gets inputted question and answers

def appendDetails(inputtedQuestion,inputtedAnswers,testFile):

    csv=[]

    # Gets inputted question and answer/answers

    question=inputtedQuestion.get(1.0,'end-1c')

    answers=inputtedAnswers.get(1.0,'end-1c')

    # Adds inputted question and answer/answers to a dictionary

    csvData={'Question':question,'Answer':answers}

    csv.append(csvData)

if len(question)!=0 and len(answers)!=0:

    saveTestData(csv,testFile,csvData,'r+')

else:

    messagebox.showerror("Questions and answers not added","Both text boxes must be filled, please try again")

# GUI to create a test

def appendTestFile(testFile):

    appendWindow=Tk()

    appendWindow.geometry("450x250")

    appendWindow.title("Create a test")

    appendWindow.configure(bg="#1EFF00")

    appendWindow.resizable(False,False)
```

```
labelTitle=Label	appendWindow,text="Add questions and answers to the test")
labelTitle.pack(side=TOP)

labelAddQuestion=Label	appendWindow,text="Add a question:",bg="#1EFF00",font=("Courier 10 underline"), borderwidth=0)
labelAddQuestion.place(x=22,y=25)

# Text widget to allow the user to enter a question
entryAddQuestion=Text	appendWindow,width=50,height=3)
entryAddQuestion.place(x=22,y=40)

labelAddAnswer=Label	appendWindow,text="Add answers:",bg="#1EFF00",font=("Courier 10 underline"), borderwidth=0)
labelAddAnswer.place(x=22,y=100)

# text widget to allow the user to enter an answer/answers
entryAddAnswer=Text	appendWindow,width=50,height=5)
entryAddAnswer.place(x=22,y=115)

# Submit button to start the process to check and save the inputted question and answer/answers
submitButton=Button	appendWindow,text="Submit",width=8,command=lambda: appendDetails(entryAddQuestion,entryAddAnswer,testFile))
submitButton.place(x=22,y=210)

# Button to display the contents of the test in a window
previewButton=Button	appendWindow,text="Preview",width=8,command=lambda: testPreview(testFile))
previewButton.place(x=120,y=210)

# Button to delete the last entry in the test file
deleteButton=Button	appendWindow,text="Delete last",width=8,command=lambda: readTestFile(testFile,"Delete"))
deleteButton.place(x=260,y=210)

# Button to start the adding grades process
finishButton=Button	appendWindow,text="Finish",width=8,command=lambda: endTest.appendWindow,testFile))
finishButton.place(x=360,y=210)

# Opens created test csv file
def openCreatedTest(testFile):
    csvFile=open(testFile,'r+')
    csvFile.write('Question,Answer\n')
```

```
file=csvFile.read()

for item in file:
    csvFile.write('{Question},{Answer}'.format(**item))
    csvFile.write('\n')
csvFile.close()

appendTestFile(testFile)

# Creates test csv file
def testFileCreator(testName,fileCreator,*gradeAppend):
    # Creates the file path for the test to be created
    csv=".csv"
    userTestName=testName.get()
    rootFile=os.path.abspath(os.curdir)
    dstFile=str(rootFile)+str("\\") + str(userTestName)+str(gradeAppend)
    dstFile=dstFile.strip('\'')
    dstFile=dstFile+str(csv)

    # Gets the list of all files on the current directory
    nameToCheck=str(userTestName)+str(csv)
    fileExists=False
    listOfFiles=os.listdir(rootFile)

    # Checks if the file path of the test to be created matches the path of one of the files on the directory. If so, then the name already checks.
    for i in range(0,len(listOfFiles)):
        file=listOfFiles[i]
        if file==nameToCheck:
            fileExists=True
            break

    rootFile=rootFile+"\\"+empty.csv"
    print(dstFile)

    # Creates the test if the file does not already exist and the input is not empty
    if userTestName!="" and fileExists==False:
        shutil.copy2(rootFile,dstFile)
```

```
fileCreator.destroy()

openCreatedTest(dstFile)

elif fileExists==True:
    messagebox.showerror("Test not created",
                         "This test already exists, please enter another name")

else:
    messagebox.showerror("Test not created",
                         "The test has not been created, please make sure all input boxes are filled and try again")

# GUI to name test file

def nameTestFile(title,function,username):
    fileCreator=Tk()
    fileCreator.geometry("450x42")
    fileCreator.title(title)
    fileCreator.configure(bg="#1EFF00")

    fileCreator.resizable(False,False)

    labelCreator=Label(fileCreator,text="Enter the name of a test:",bg="#1EFF00",font=("Courier 10"),borderwidth=0)
    labelCreator.place(x=22,y=5)

    # Entry to allow the user to enter the test name
    nameOfTestBox=Entry(fileCreator,width=52)
    nameOfTestBox.place(x=22,y=20)

    # Changes function of the submit button based on the arguments in the function call
    if function=="Create":
        submitButton=Button(fileCreator,text="Submit",command=lambda:testFileCreator(nameOfTestBox,fileCreator))
        submitButton.place(x=375,y=13)
```

```
elif function=="Complete":  
    submitButton=Button(fileCreator,text="Submit",command=lambda:checkFileName(nameOfFileBox,username))  
    submitButton.place(x=375,y=13)  
  
#  
# SECONDARY MENU  
#  
  
# Creates the secondary menu GUI  
def secondaryMenu(username,adminStatus):  
    sec=Tk()  
    sec.geometry("450x50")  
    sec.title("Features Menu")  
    sec.configure(bg="#2019F7")  
  
    sec.resizable(False,False)  
  
    labelSecMenu=Label(sec,text="Welcome to the program")  
    labelSecMenu.pack(side=TOP)  
  
    buttonComplete=Button(sec,text="Complete a test",command=lambda: nameTestFile("Complete a test","Complete",username))  
    buttonComplete.place(x=10, y=20)  
  
    buttonRFeedback=Button(sec,text="Recieve feedback",command=lambda: getUsersAndTestNames("Recieve",username))  
    buttonRFeedback.place(x=330,y=20)  
  
    # Defines additional buttons if the account logged in has admin status  
    if adminStatus==True:  
        sec.geometry("450x200")  
        buttonGFeedback=Button(sec,text="Give feedback",command=lambda: feedbackSelectorGUI())  
        buttonGFeedback.place(x=330,y=150)  
  
        buttonCreate=Button(sec,text="Create a test",command=lambda: nameTestFile("Create a test","Create",username))  
        buttonCreate.place(x=10,y=150)
```

```
# Opens login csv file

def openLogin(buttonOutput):
    csvFile=open('login.csv')
    csvFileContent=(csvFile)
    next(csvFile)

    databaseList=[]

    # Reads the contents of the 'login.csv' file into a dictionary
    for row in csvFileContent:
        database=row.strip().split(",")
        heading=['Username','Password','Admin']
        data=zip(heading,database)
        DataDict=dict(data)
        databaseList.append(DataDict)

    # Starts either the login or registration process as this function is assigned to both buttons
    if buttonOutput=="startLogin":
        loginFunction(databaseList)
    elif buttonOutput=="startRegister":
        registerFunction()

    #
    # REGISTRATION
    #

# Gets inputted username and password and checks if is valid

def registerAccount(inputtedUsername,inputtedPassword,inputtedAdmin):
    login=[]
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    admin=inputtedAdmin.get()

    csvFile=open('login.csv','r+')
    csvReader=csv.reader(csvFile)
    rows=list(csvReader)
    csvFile.close()
```

```
usernameTaken=False

for i in range(1,len(rows)):
    if username==rows[i][0]:
        usernameTaken=True
        break

# Checks if the inputted username is not blank and does not already exist, and checks if the inputted password is not blank. If these conditions are not met a relevant error message is displayed.

if username!="" and password!="" and usernameTaken==False:
    logindata={'Username':username,'Password':password,'Admin':admin}
    login.append(logindata)

    # Saves login details if the inputted login details are valid
    saveLoginData(login)

elif usernameTaken==True:
    messagebox.showerror("Username already exists","Username already exists, please enter a new username")
else:
    messagebox.showerror("Registration unsuccessful","Registration unsuccessful, please try again")

# Saves inputted username and password

def saveLoginData(login):
    try:
        fileHandle=open('login.csv','r+')
        fileContent=fileHandle.read()

        # Adds headings to the file if they don't already exist
        if fileContent.strip()=='':
            fileHandle.write('Username,Password,Admin\n')

        # Writes the new login details using the necessary feedback
        for item in login:
            fileHandle.write('{Username},{Password},{Admin}'.format(**item))
            fileHandle.write('\n')

        fileHandle.close()

        messagebox.showinfo("Registration successful","Login details saved")

    except OSError:
        print('Can\'t write to file')
```

```
# GUI for account registration

def registerFunction():

    registerWindow=Tk()

    registerWindow.geometry("450x200")

    registerWindow.title("Registration")

    registerWindow.configure(bg="#FF2929")

    labelRegistration=Label(registerWindow,text="Registration")

    labelRegistration.pack(side=TOP)

    registerWindow.resizable(False,False)

    labelUsername=Label(registerWindow, text="Username:",bg="#FF2929",font=("Courier 10 underline"), borderwidth=0)

    labelUsername.place(x=20,y=20)

    # Entry to allow the user to enter a username

    usernameBox=Entry(registerWindow,width=67)

    usernameBox.place(x=20, y=40)

    labelPassword=Label(registerWindow, text="Password:",bg="#FF2929",font=("Courier 10 underline"), borderwidth=0)

    labelPassword.place(x=20,y=80)

    # Entry to allow the user to enter a password

    passwordBox=Entry(registerWindow,width=67,show="*")

    passwordBox.place(x=20,y=100)

    labelAdmin=Label(registerWindow,text="Admin code:",bg="#FF2929",font=("Courier 10 underline"), borderwidth=0)

    labelAdmin.place(x=20,y=130)

    # Entry to allow the user to enter an admin code

    adminBox=Entry(registerWindow,width=20)

    adminBox.place(x=20,y=150)

    # Button to check and save the inputted login details

    buttonRegister=Button(registerWindow,text="Register",command=lambda :registerAccount(usernameBox,passwordBox,adminBox))

    buttonRegister.place(x=300,y=150)
```

```
#  
# LOGIN  
  
# Login GUI  
  
def loginFunction(databaseList):  
    loginWindow=Tk()  
    loginWindow.geometry("450x200")  
    loginWindow.title("Login")  
    loginWindow.configure(bg="#FFA21F")  
  
    labelLogin=Label(loginWindow,text="Login")  
    labelLogin.pack(side=TOP)  
  
    loginWindow.resizable(False,False)  
  
    labelUsername=Label(loginWindow, text="Username:",bg="#FFA21F",font=("Courier 10 underline"), borderwidth=0)  
    labelUsername.place(x=20,y=20)  
  
    # Entry to allow the user to enter a username  
    usernameBox=Entry(loginWindow,width=67)  
    usernameBox.place(x=20, y=40)  
  
    labelPassword=Label(loginWindow, text="Password:",bg="#FFA21F",font=("Courier 10 underline"), borderwidth=0)  
    labelPassword.place(x=20,y=80)  
  
    # Entry to allow the user to enter a password  
    passwordBox=Entry(loginWindow,width=67,show="*")  
    passwordBox.place(x=20,y=100)  
  
    # Button to check the inputted login details  
    buttonLogin=Button(loginWindow,text="Login",command=lambda :loginAccount(usernameBox,passwordBox,loginWindow,databaseList))  
    buttonLogin.pack(side=BOTTOM, pady=10)  
  
    # Gets inputted login details and checks them
```

```
def loginAccount(inputtedUsername,inputtedPassword,loginWindow,databaseList):
    username=inputtedUsername.get()
    password=inputtedPassword.get()
    loginStatus=False
    adminStatus=False
    for i in databaseList:
        # Checks if the inputted login details are correct
        if i['Username']==username and i['Password']==password:
            messagebox.showinfo("Login successful","Login successful")
            loginStatus=True
            loginWindow.destroy()
        # Opens secondary menu with admin features if the registered admin code is correct
        if i['Admin']=='testadmin':
            adminStatus=True
    if loginStatus==True:
        secondaryMenu(username,adminStatus)
    else:
        messagebox.showerror("Login attempt failed","Login attempt failed, please try again")
```

```
#  
# GUIDE  
#
```

```
# Creates window with 'guide.txt's contents
```

```
def openGuide():
    guideWindow=Tk()
    guideWindow.geometry("450x200")
    guideWindow.title("Guide")
    guideWindow.configure(bg="#EAFF00")

    guideWindow.resizable(False,False)
```

```
    textFile=open('guide.txt')
    text=textFile.read()
```

```
    scrollbar=Scrollbar(guideWindow)
    scrollbar.pack(side=RIGHT,fill=Y)
```

```
# Text widget to insert the contents of the 'guide.txt' file into the window so that it can be displayed
textGuide=Text(guideWindow,bg="#EAF00", width=450,wrap=WORD)
textGuide.pack()

textGuide.insert(END,text)

textGuide.config(yscrollcommand=scrollbar.set,state=DISABLED)
scrollbar.config(command=textGuide.yview)

textFile.close()

#
# MAIN MENU BUTTONS
#
# Main menu buttons are defined and the window is opened

buttonRegistration=Button(root,text="Register a new\n account",command=lambda: openLogin("startRegister"))
buttonRegistration.place(x=10, y=20)

buttonLogin=Button(root,text="Login with an\n exisiting account",command=lambda: openLogin("startLogin"))

buttonLogin.place(x=330,y=20)

buttonEnd=Button(root,text="Exit the program",command=lambda: os._exit(0))
buttonEnd.place(x=330,y=150)

buttonGuide=Button(root,text="Guide on how to use this program",bg="#33BFFF",font=("Courier 10 underline"), borderwidth=0, command=openGuide)
buttonGuide.place(x=10,y=150)

root.mainloop()
```