Software Engineering Assignment - 2

Features

Created a Registration GUI using the Python Module Tkinter in an Object-oriented fashion. Features include but are not limited to-

- **User Registration**
  - Three Kinds of users (Professor, PG student, UG student) following a Class hierarchy as described on moodle.
  - Different classes consist of different attributes while the common attributes are to be implemented via Inheritance, For example a Person Class can be the parent class of all while the student class can be a parent to the PG and UG classes.
- **User Login**
  - User login based on UID (Unique ID), Password and username. Passwords must follow the specified format.
  - Three incorrect login attempts should lead to a deregistration.
- **Deregistration of an account**
  - Users should authenticate themselves in order to deregister their account.
- **Pickle file to store User data**
  - Python pickle module used to convert and store user data in the form of binary data.
  - Error handling of the case when the pickle file is unavailable program must create a new file and write to it.
- **Editing data**
  - Users have the option to edit data post login.
  - The password and email mandatorily still abide by the standards set while registration.

Code Documentation

Implementation and explanation of archetype functions and classes along with their usage follow.

- **Class Application**

Unset

```
window.grid_rowconfigure(0, minsize=1000)
window.grid_columnconfigure(0, minsize=1000)
```

Configures Window size of Tkinter Application. It should be noted that size configurations depends on the resolution of the device screen being used.

Unset

```
self.frames = {}
        for F in (Login, ProfReg, PGReg, RegType, UGReg, StuReg, DeReg):
            frame = F(window, self)
```

```
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky="nsew")
```

Registers the app pages to the application class to be rendered and sets the location of the page to be rendered.

Unset

```
def show_frame(self, page):
        frame = self.frames[page]
        frame.tkraise()
```

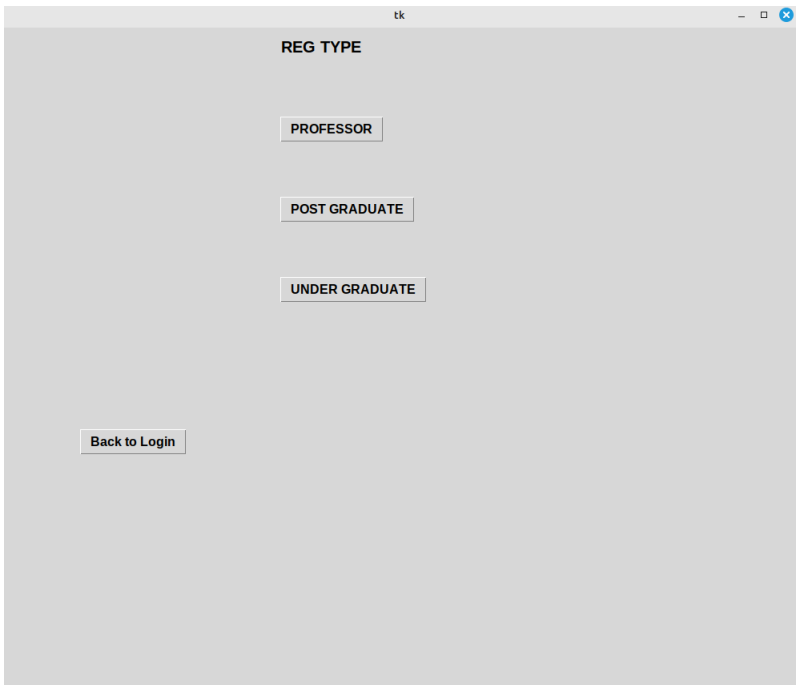
Tkraise method will raise a given frame above all others, essentially change the screen on display

- **Class RegType**

Unset

```
def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        self.TITLE = tk.Label(self, text="REG TYPE", font=("Arial", 15, "bold"))
        self.TITLE.place(x=350, y=10)

        self.PROF_BUTTON = tk.Button(
            self,
            text="PROFESSOR",
            font=("Arial", 12, "bold"),
            command=lambda: controller.show_frame(ProfReg),
        )
        self.PROF_BUTTON.place(x=350, y=110)
```

self.title sets the page title as "REG TYPE". As an example, The PROF_BUTTON attribute shows frame "ProfReg" upon clicking.



- **Class PersonReg**

Unset

```
class PersonReg(tk.Frame):
   def __init__(self, parent, controller):
       tk.Frame.__init__(self, parent)

       self.AttributesList = []  ## USED FOR CLEARING OF LABELS NEXT TIME A PAGE IS
                                     RENDERED
       self.NAME_PLACE = tk.Label(self, text="Name: ", font=("Arial", 12, "bold"))
       self.NAME_PLACE.place(x=200, y=100)  ## places label on screen
       self.NAME = tk.Entry(self, width=30, bd=5)
       self.AttributesList.append(self.NAME)
       self.NAME.place(x=400, y=100)
```

PersonReg is the parent class to all user types. It's has some of the key User parameters like
NAME, Password(PW and PW2), website (WEBPAGE), Email (EMAIL). The other attribute codes are much similar to NAME and have not been shown here.

Unset

```
def PW_verification(self):
       password_pattern = re.compile(
           r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%&*])[^\s]{8,12}$"
       )
```

Class method uses the given Regex to verify the password comment in the lines that follow.

Unset

```
messagebox.showwarning("PASSWORD ERROR", "Passwords do not match!")
```

PW_verifiaction also matches PW and PW2 and displays the following error box if they do not match. Error boxes are displayed via the "Tkinter messagebox".

Unset

```
def email_verification(self):
       pattern = r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"
       if re.match(pattern, self.EMAIL.get()) is not None:
           return True

       else:
           messagebox.showwarning("Invalid Email", "Please enter a valid Email")
           return False
```

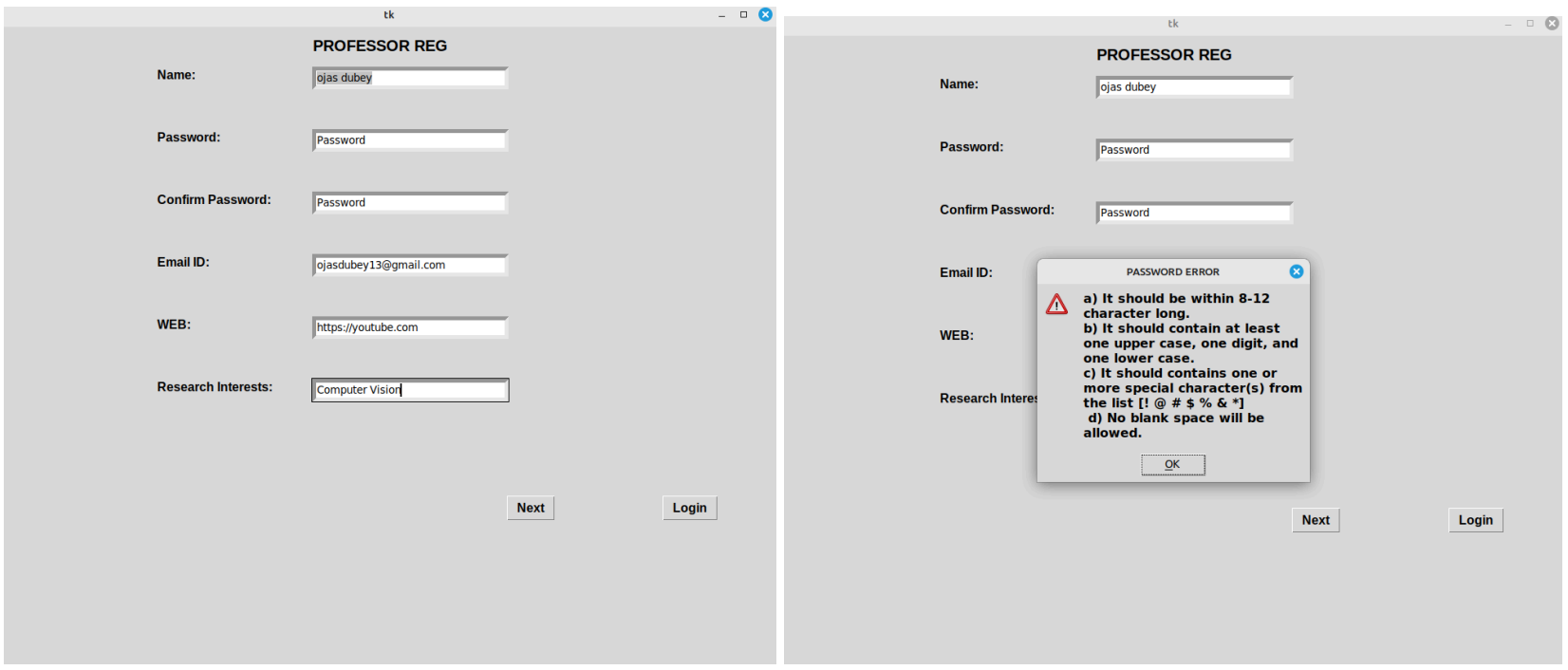
Much similarly, email_verification uses this regex to check email format.

- **Class ProfReg**

Unset

```
class ProfReg(PersonReg):
   def __init__(self, parent, controller):
       super().__init__(parent, controller)
```

ProfReg inherits from the PersonReg class. super().__init__ initializes the parameters of the parent class in the child, although child class can overwrite them by redefining the attribute/method.
This class has an added attribute of RESEARCH, which is to be filled for research areas of the professor.



Images showing professor registration page and display of error if Regex is not matched.

- **Function to edit the user database**

Unset

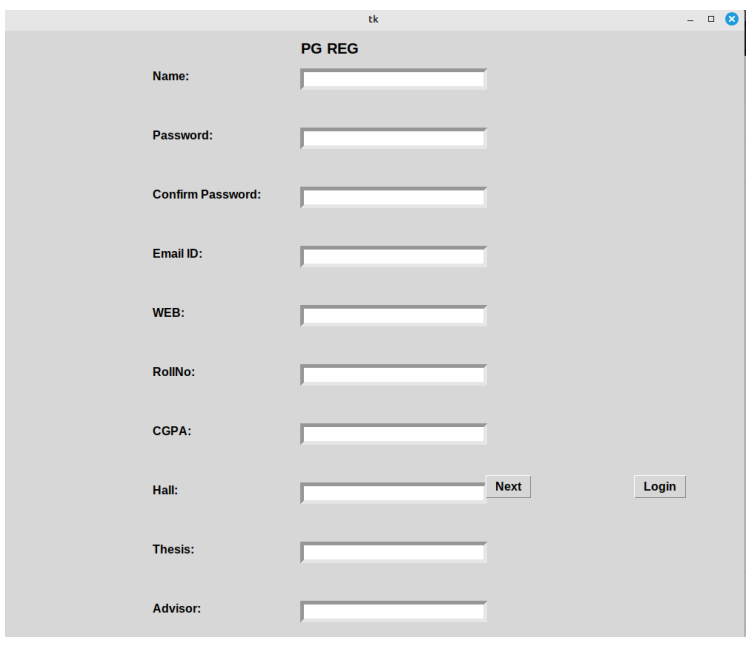
```
def register(self, parent, controller):
       if not self.PW_verification() or not self.email_verification():
           return

       uid = datetime.now().strftime("%S.%f")[-6:]

       try:
           dbfile = open("database.pkl", "rb")
           db = pickle.load(dbfile)

       except:
           dbfile = open("database.pkl", "wb")
           db = {}
       db[uid] = {
           "name": self.NAME.get(),
           "password": self.PW.get(),
           "email": self.EMAIL.get(),
           "WEB": self.WEB.get(),
           "research": self.RESEARCH.get(),
       }
       dbfile.close()

       dbfile = open("database.pkl", "wb")
       pickle.dump(db, dbfile)
```

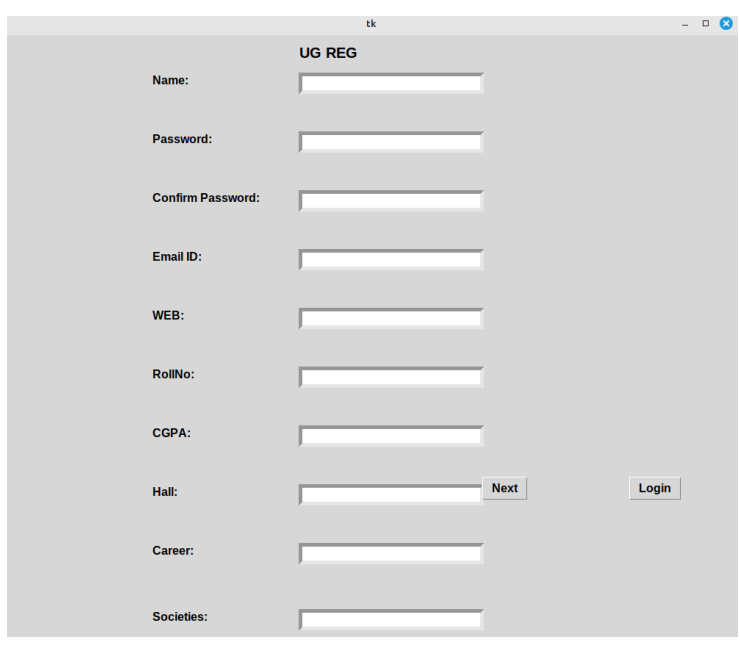
```
        print(db)

        dbfile.close()

        for item in self.AttributesList:
            item.delete(0, tk.END)

        frame = Home(parent, uid, controller)
        frame.grid(row=0, column=0, sticky="nsew")
        frame.tkraise()
```

This method is recurrently used throughout the program. It essentially loads the pickle database as a dictionary and adds the current user data to it. It then proceeds to write the updated data back to the file.
Initially, A UID assigned to each user is created based on the microsecond timestamp such that the chance of collisions is very low. "datetime" module was used for the current timestamp.
Error handling has been implemented for the case if the file does not exist, A new file named "database.pkl" should be created.
Similar methods are used in each of the classes of registration and also used when user data is to be deleted. It may not be described again as all functionality is similar.

- **Classes StuReg, PGReg, UGReg**

  Classes UGReg and PGReg inherit from StuReg and have a few additional attributes.
  They are overall similar in functionality to the prof registration page, hence only images of theirs are being attached.





  Login button takes the user back to the login page.

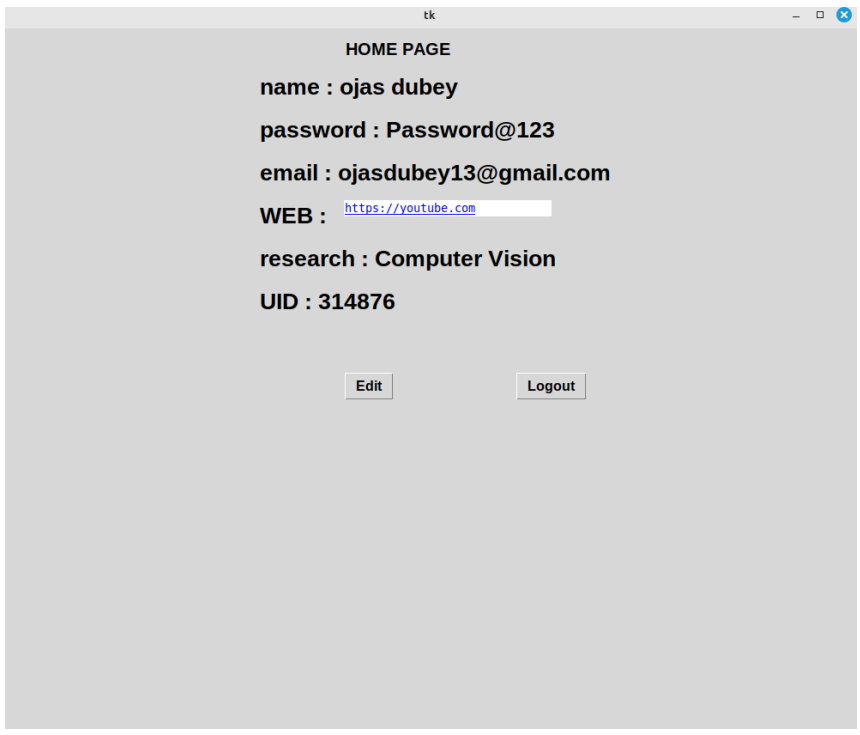
- **Class Home**

Unset

```
data = pickle.load(dbfile)
        if uid in data.keys():
            for key, value in data[uid].items():
                if key != "WEB":
                    ELEMENT = tk.Label(
                        self, text=f"{key} : {value}", font=("Arial", 20, "bold")
```

```
                )
                ELEMENT.place(x=300, y=y)

            else:
                ELEMENT = tk.Label(
                    self, text=f"{key} : ", font=("Arial", 20, "bold")
                )
                ELEMENT.place(x=300, y=y)

                self.link_text = tk.Text(
                    self,
                    height=1,
                    width=30,
                    wrap="none",
                    borderwidth=0,
                    highlightthickness=0,
                )
                self.link_text.insert("1.0", value)
                self.link_text.tag_configure(
                    "hyperlink", foreground="blue", underline=True
                )
                self.link_text.tag_add("hyperlink", "1.0", "1.end")
                self.link_text.tag_bind(
                    "hyperlink",
                    "<Button-1>",
                    self.open_link,
                )
                self.link_text.place(x=400, y=y)

            y = y + 50

        ELEMENT = tk.Label(self, text=f"UID : {uid}", font=("Arial", 20, "bold"))
        ELEMENT.place(x=300, y=y)
        y = y + 100
```

This codeblock explains the loading and display of the database.
Barring the case of the hyperlink (covered ahead), A simple "Label" is implemented which creates a {key} : {value} display as in the f-string.
For the hyperlink, a clickable link is inserted with blue underlined text which is opened using the "webbrowser" module.



Logout button takes the user back to the login page**.**

- **Class Edit**

Unset

```
self.attributes[key] = tk.Entry(self, width=30, bd=5)
                self.attributes[key].place(x=550, y=y)
                self.attributes[key].insert(1, value)
```

Instead of having labels for values like Home, This class uses the "Entry" box and places relevant values in the box. This makes the content editable by the user.

Unset

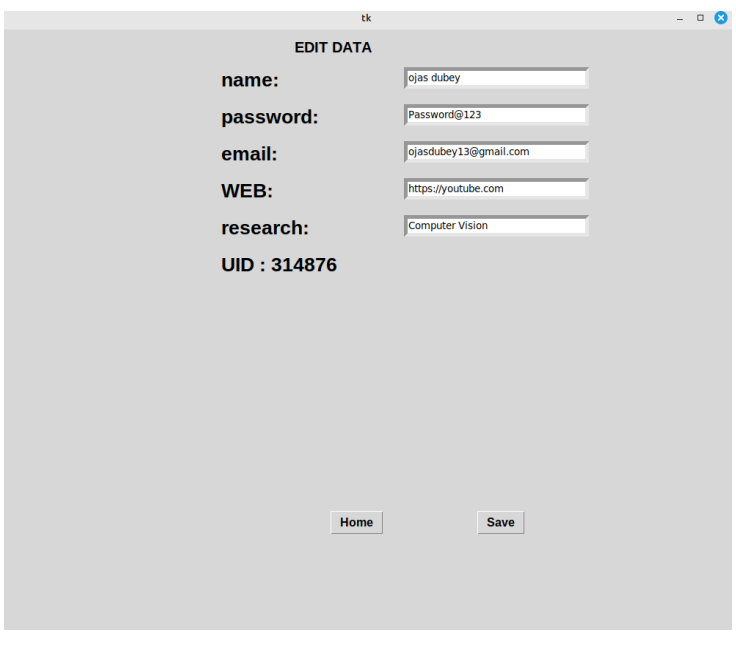
```
def render_home(self, parent, uid, controller):
        frame = Home(parent, uid, controller)
        frame.grid(row=0, column=0, sticky="nsew")
        frame.tkraise()
```

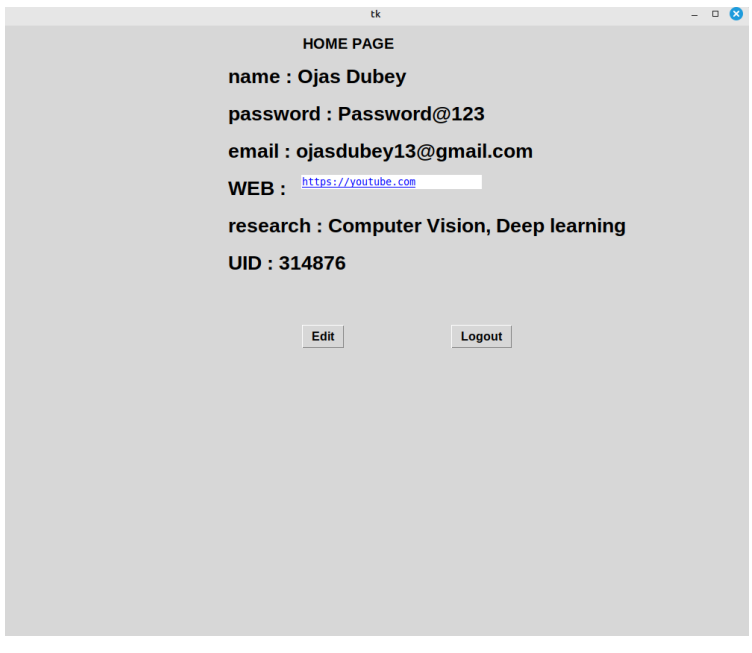
Method to render Homepage in case of an event of the button click. tkraise() is used again for this purpose as it is throughout the code.

Unset

```
def save_attributes(self, uid, parent, controller):
```

Regex methods used in the Registration page are used again here to ensure that users do not edit their passwords or email to an invalid.





- **Class DeReg**

Unset

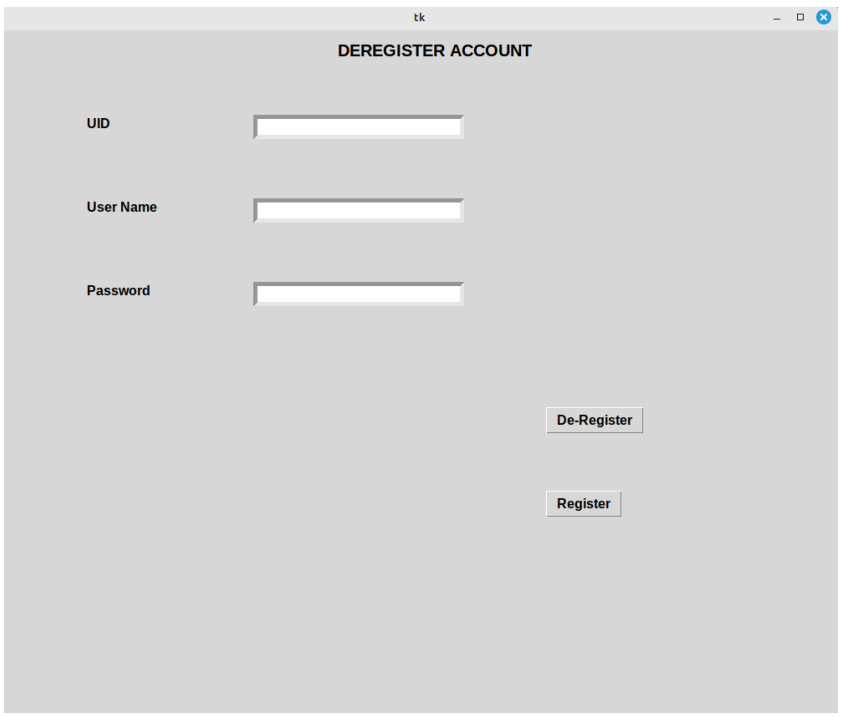
```
def PWCheck(self, uid, PW, name, parent, controller):
        dbfile = open("database.pkl", "rb")
        try:
            data = pickle.load(dbfile)

            if uid in list(data.keys()):
                if data[uid]["password"] == PW and data[uid]["name"] == name:
                    self.DeRegCheck(controller, parent, uid)

                else:
                    messagebox.showwarning("CREDENTIALS ERROR", "Credential
Mismatch!")
        except EOFError:
            return

        dbfile.close()
```

Method checks whether name and password match with the given pickle database.

Unset

```
def DeRegCheck(self, controller, parent, uid):
        dbfile = open("database.pkl", "rb")
        try:
            data = pickle.load(dbfile)

            if uid in list(data.keys()):
                del data[uid]

        except EOFError:
            return

        print("here", uid)

        dbfile.close()

        dbfile = open("database.pkl", "wb")

        if len(data) == 0:
            dbfile.close()

        else:
            pickle.dump(data, dbfile)
            dbfile.close()

        controller.show_frame(Login)
```

Method finds the user in the database and uses python dictionary "del" keyword to logically delete the user data from the pickle database.



- **Class Login**

Unset

```
class Login(DeReg):
   def __init__(self, parent, controller):
       tk.Frame.__init__(self, parent)
```

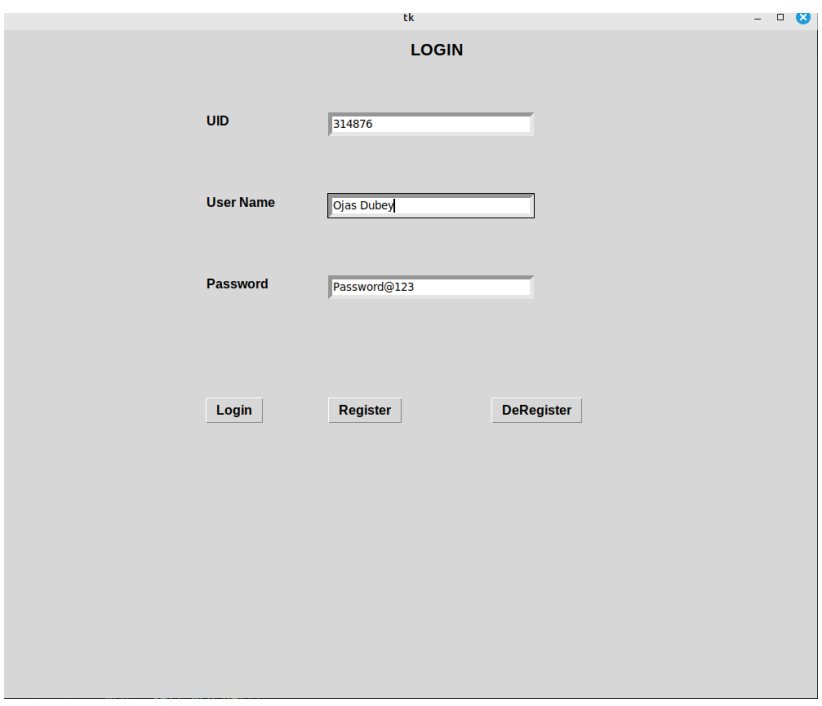
Inherits placement and methods from the DeReg class. This page is the first to be rendered upon loading.
Inherits from DeReg because of the clause of user deregistration upon 3 unsuccessful attempts for registration.

Unset

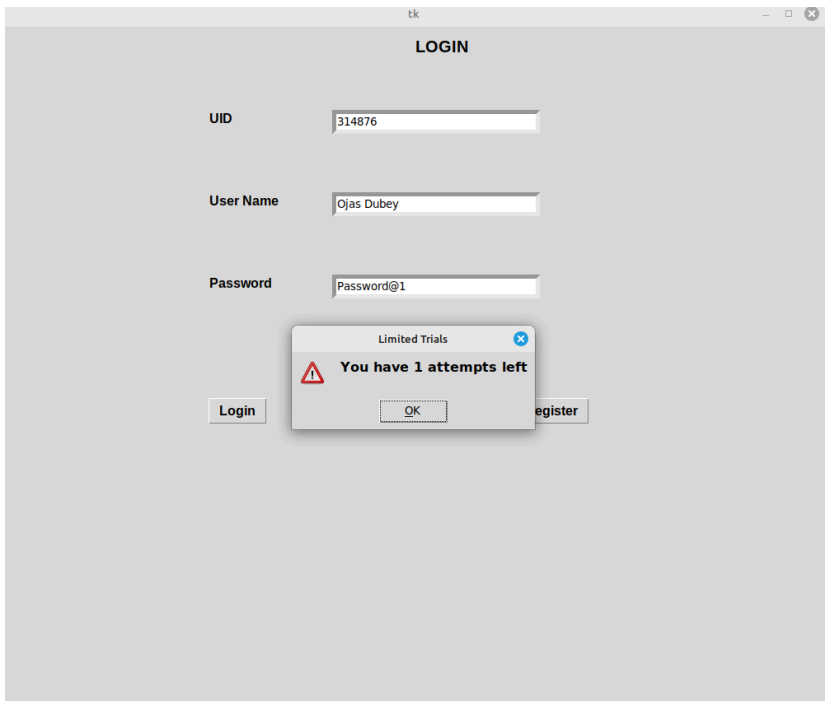
```
def LoginCheck(self, parent, controller):
       boolean = False
       UIDExists = False

       try:
           dbfile = open("database.pkl", "rb")
           data = pickle.load(dbfile)  ## LOADS DATA

       except:
           dbfile = open("database.pkl", "wb")
           data = {}

   print(data)   ## PRINTS DATABASE TO THE CONSOLE AT EACH ATTEMPT

   if self.UID.get() in list(data.keys()):
       UIDExists = True

       if (
           data[self.UID.get()]["name"] == self.Name.get()
           and data[self.UID.get()]["password"] == self.PW.get()
       ):
           boolean = True  ## CHECK CORRECTNESS OF NAME AND PW
       else:
           boolean = False
           print("[INCORRECT INPUT]")
```

```
    dbfile.close()

    if not UIDExists:
        messagebox.showwarning(
            "LOGIN ERROR", "USER ID DOES NOT EXIST, PLEASE REGISTER"
        )
        controller.show_frame(Login)   ## DATA EVALUATED CORRECTLY
    else:
        if boolean:
            print(self.UID.get())
            frame = Home(parent, self.UID.get(), controller)
            frame.grid(row=0, column=0, sticky="nsew")
            frame.tkraise()
        else:
            self.strike = self.strike - 1  ## ONE CHANCE REDUCED
            if self.strike > 0:
                messagebox.showwarning(
                    "Limited Trials", f"You have {self.strike} attempts left"
                )

            if self.strike <= 0:
                print("[TOO MANY ATTEMPTS , USER DEREGISTERED")
                messagebox.showwarning(
                    "Limited Trials", f"[TOO MANY ATTEMPTS , USER DEREGISTERED"
                )
                self.DeRegCheck(controller, parent, self.UID.get())
## USER DEREGISTERED

                controller.show_frame(RegType)
            return

        for item in self.AttributesList:
            item.delete(0, tk.END)

        return
```

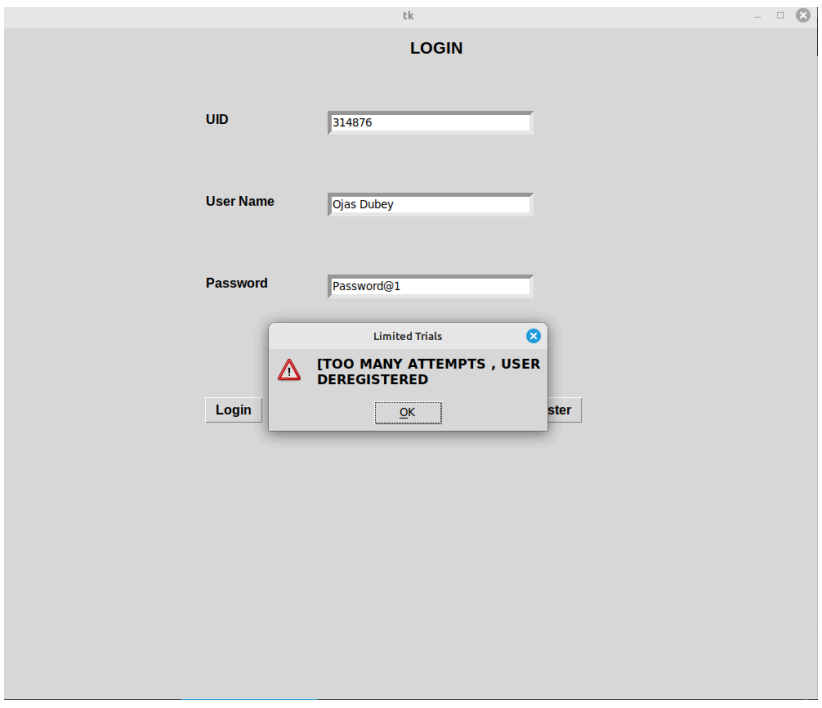
This method implements the entire login logic.
UIDExists checks whether the UID exists in the database. boolean will evaluate to true in case the credentials match precisely, in which case the Home page is rendered.
self.strike reduces by 1 each time boolean evaluates to false.
self.UID.get() method fetches whatever has been in the Input box on the screen.
Warning messages are flashed at each unsuccessful attempt. But in case UID is not entered or not found, That does not count as a strike.
Uses DeRegCheck method to delete user logically in case of 3 incorrect trials.



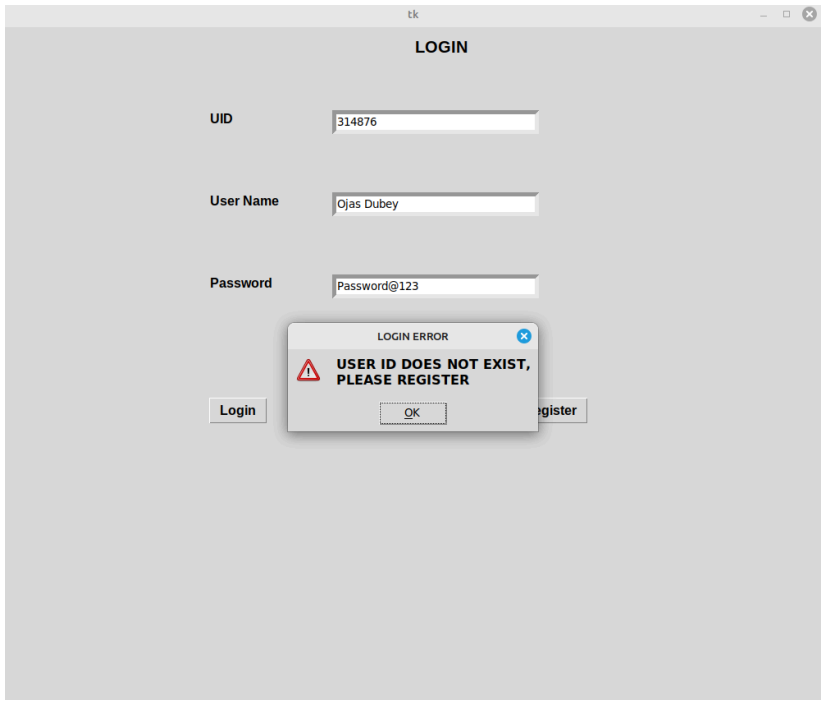
Correct user data



Incorrect attempt



User deregistered



UID not found again even if credentials are valid