



FEU Institute of Technology

Department of Information Technology

IT0011: INTEGRATIVE PROGRAMMING AND TECHNOLOGIES

SECTION

Final Project



Grade

Submitted by:



BACCAY, SHERYLL ANNE
MARIE D.



COLLO, PAUL BENEDICT
V.



CUETO, ALEXA JOYCE
G.



PUA, CHARLES MICHAEL
C.

Submitted to:

MR. JOSEPH CALLEJA

DOCUMENT CONTENTS:

I. INTRODUCTION

The User Registration Program is a Python-based application designed to streamline user management through its several features allowing users to sign up, view registered records, and search a specific record. This program offers a user-friendly interface and simple yet effective user management through a graphical user interface (GUI) built with Tkinter. It also features data persistence using file handling operations through CSV files that stores, displays, and retrieves user information together with error-trapping mechanisms for error handling and input validations to ensure proper data format and enhance structure, flow, and efficiency of the system. In addition, other concepts under the Python programming language were also incorporated to implement a lightweight and efficient user registration system. This program can also be used in different fields in the workplace with related user management needs, such as for student enrollment systems, employee records, customer registrations, and small business operations. Overall, this system was created to improve the understanding of how useful the Python programming language is in solving and offering effective solutions to real-world problems.

II. OBJECTIVES

The objective of our User Registration System is to provide a seamless and easy way to store information for its users. Structured with a CRUD, our program can handle the basic functionalities required by the system such as:

- Creating records of each unique user.
- Displaying the information of all persons registered in the system.
- Manually searching for a specific record by simply typing in their last name.
- Arrange the records by last name in alphabetical order to offer optimal organization.
- Update a record's registered information.
- Saved into a separate file automatically.
- Remove specific records from the file.

By integrating past key concepts we have successfully created such a system. By using GUI, our system came alive from a simple console into an interactive interface that encourages fun and optimism from our users.

III. PROGRAM SCREEN SHOTS

Capture the program window for each process that has been taken. Write some description about the captured screen.

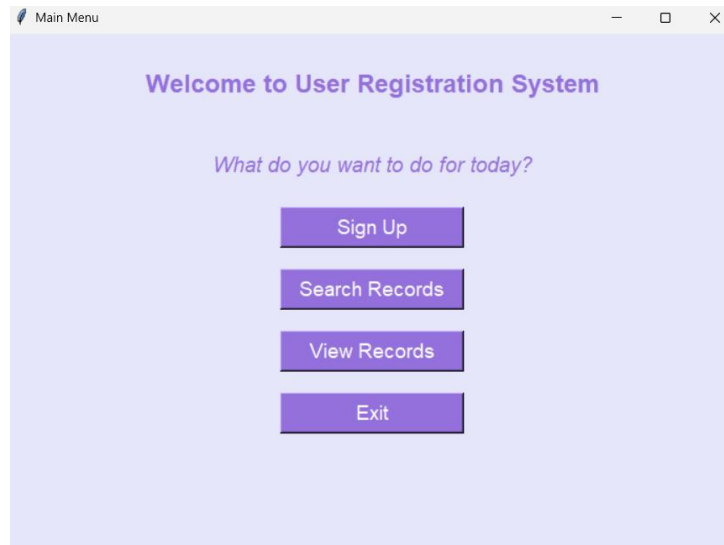


Figure 1. Main Program

Figure 1 shows the Main Menu of the program consisting tabs for the functions: Sign Up, Search Records, View Records, and Exit respectively.

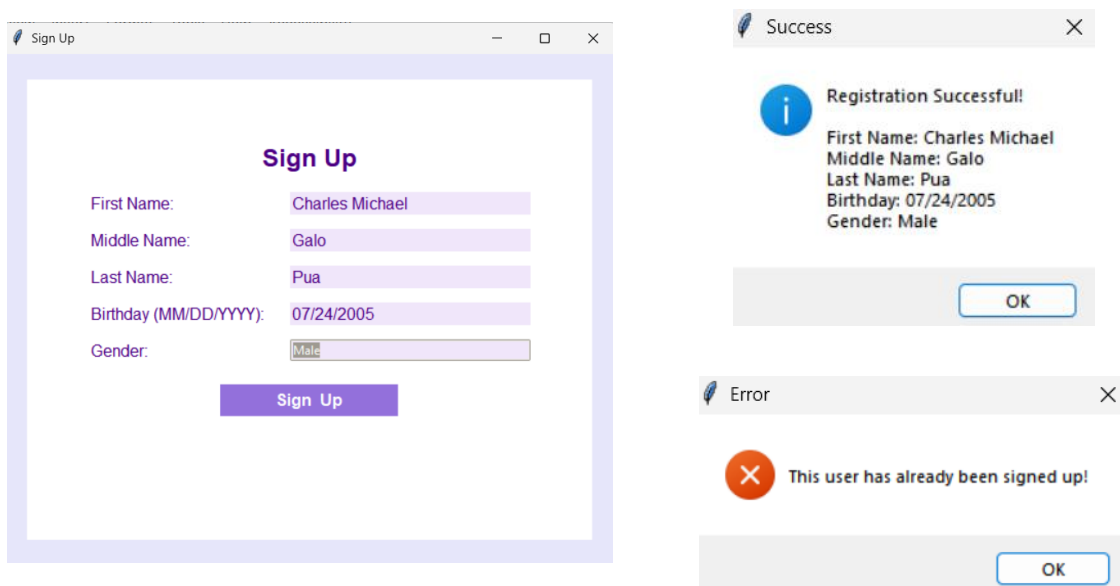
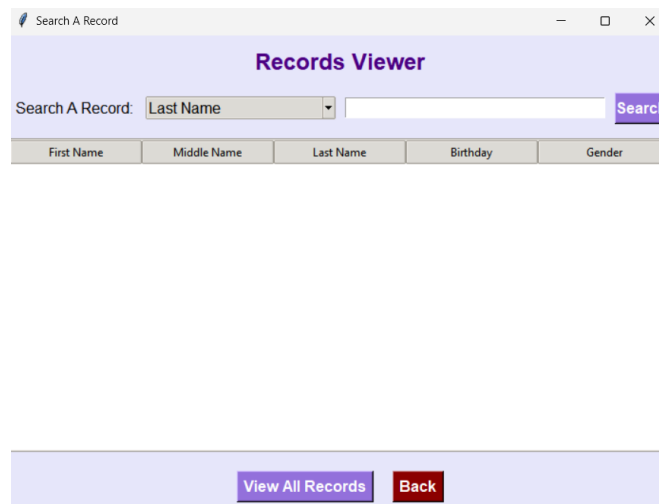


Figure 2. Sign Up

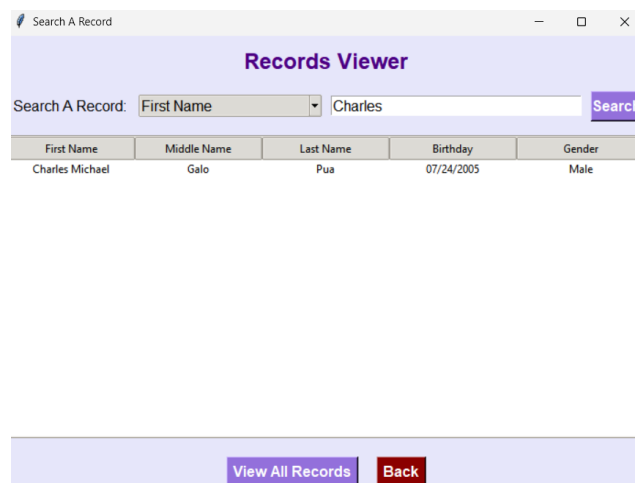
Figure 2 shows the Sign Up page of the program. It prompts the user a form with the following fields: first, middle, and last name, birthday, and gender. Once done, a message window shows that the registration has been successful. Otherwise, an error message pops up when a duplicate input is detected



The screenshot shows a window titled "Search A Record" with a subtitle "Records Viewer". Below the subtitle, there is a search bar with a dropdown menu set to "Last Name" and a text input field. A "Search" button is to the right of the input field. Below the search bar is a table with five columns: "First Name", "Middle Name", "Last Name", "Birthday", and "Gender". The table is currently empty. At the bottom of the window, there are two buttons: "View All Records" and "Back".

Figure 3. Records Viewer

Figure 2 shows the Records Viewer page of the program. The user can either search a record by first name, middle name, last name, or simply view all records.



The screenshot shows the same "Search A Record" window with "Records Viewer" subtitle. The search bar dropdown is now set to "First Name" and the text input field contains "Charles". The "Search" button is highlighted. Below the search bar, the table now displays one record:

First Name	Middle Name	Last Name	Birthday	Gender
Charles Michael	Galo	Pua	07/24/2005	Male

At the bottom of the window, the "View All Records" and "Back" buttons are still present.

Figure 3.1 Records Viewer - Search by First Name

Figure 3.1 shows the function of searching a record by first name.

The screenshot shows a web application window titled "Search A Record". The main heading is "Records Viewer". Below the heading, there is a search form with a dropdown menu labeled "Search A Record:" and a text input field. The dropdown menu is set to "Middle Name" and the text input field contains "Galo". A "Search" button is to the right of the input field. Below the search form, there is a table with the following data:

First Name	Middle Name	Last Name	Birthday	Gender
Charles Michael	Galo	Pua	07/24/2004	Male

At the bottom of the window, there are two buttons: "View All Records" and "Back".

Figure 3.2 Records Viewer - Search by Middle Name

Figure 3.1 shows the function of searching a record by first name.

The screenshot shows the same web application window as Figure 3.1, but with the dropdown menu set to "Last Name" and the text input field containing "Pua". The table below the search form shows the same data as in Figure 3.1:

First Name	Middle Name	Last Name	Birthday	Gender
Charles Michael	Galo	Pua	07/24/2005	Male

At the bottom of the window, there are two buttons: "View All Records" and "Back".

Figure 3.3 Records Viewer - Search by Last Name

Figure 3.2 shows the function of searching a record by last name.

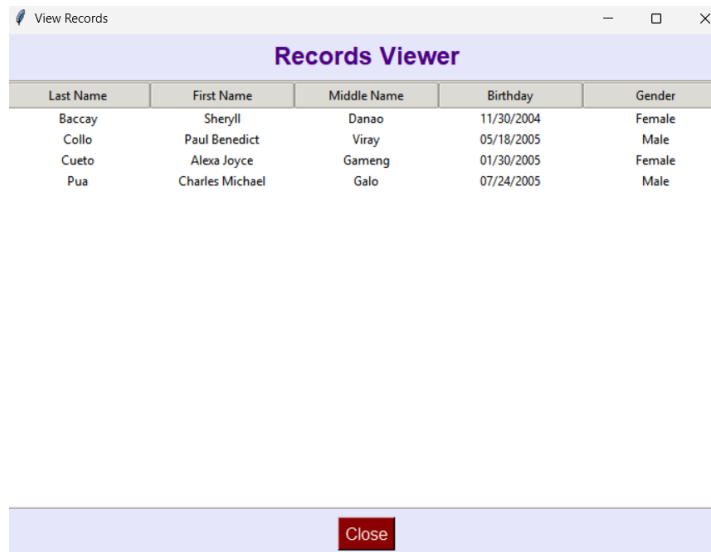


Figure 4. View Records

Figure 4 shows the function of “View All Records”.

IV. SOURCE CODE

#main.py

```
import tkinter as tk #gui
from signUp import signUpWindow #signUp.py
from searchRecord import searchRecordWindow #searchRecord.py
from viewRecord import viewRecordWindow #viewRecord.py

def main_menu():
    root = tk.Tk()
    root.title("Main Menu")
    root.geometry("700x500") #gemoetry
    root.configure(bg="#e6e6fa")

    tk.Label(root, text="Welcome to User Registration System",
             font=("Roboto", 18, 'bold'), bg="#e6e6fa", fg="#9370DB").pack(pady=30)

    tk.Label(root, text="What do you want to do for today?", font =
             ("Roboto", 15, 'italic'), bg="#e6e6fa", fg="#9370DB").pack(pady= 15)

    button_config = {"font": ("Roboto", 14), "bg": "#9370DB", "fg": "white",
                     "width": 15, "height": 1}
```

```

tk.Button(root, text="Sign Up", command=lambda: signUpWindow(root),
**button_config).pack(pady=10)

tk.Button(root, text="Search Records", command=lambda:
searchRecordWindow(root), **button_config).pack(pady=10)

tk.Button(root, text="View Records", command=lambda:
viewRecordWindow(root), **button_config).pack(pady=10)
tk.Button(root, text="Exit", command=root.quit,
**button_config).pack(pady=10)

root.mainloop()

if __name__ == "__main__":
    main_menu()

```

#signUp.py

```

import tkinter as tk #gui
from tkinter import messagebox, ttk
from datetime import datetime
import csv #file handling/data persistence
from fileHandler import saveRecords #csv file

def isUserExists(firstName, middleName, lastName):
    try:
        with open("records.csv", "r", newline="") as file:
            reader = csv.reader(file)
            for row in reader:
                if len(row) >= 3 and row[0] == firstName and row[1] ==
middleName and row[2] == lastName:
                    return True #User already exists
    except FileNotFoundError:
        return False #If file doesn't exist, no duplicates
    return False #No match found

def signUpWindow(mainWindow):
    root = tk.Toplevel(mainWindow) #toplevel
    root.title("Sign Up")
    root.geometry("700x500")
    root.configure(bg='#e6e6fa')

    #Function to handle signup
    def signUp():
        firstName = firstNameEntry.get().strip()
        middleName = middleNameEntry.get().strip()
        lastName = lastNameEntry.get().strip()
        birthday = birthdayEntry.get().strip()

```

```

gender = genderEntry.get().strip()

if not firstName or not middleName or not lastName or not birthday or
not gender:
    messagebox.showerror("Error", "This field is required. Please
fill in required inputs.")
    return

try:
    datetime.strptime(birthday, "%m/%d/%Y")
except ValueError:
    messagebox.showerror("Error", "Birthday must be in the format of
MM/DD/YYYY.")
    return

if gender not in ["Male", "Female", "Other"]:
    messagebox.showerror("Error", "Please input a valid gender.")
    return

if isUserExists(firstName, middleName, lastName):
    messagebox.showerror("Error", "This user has already been signed
up!")
    return (signUpWindow)

messagebox.showinfo("Success", f"Registration Successful!\n\nFirst
Name: {firstName}\nMiddle Name: {middleName}\nLast Name:
{lastName}\nBirthday: {birthday}\nGender: {gender}")

saveRecords("records.csv", [firstName, middleName, lastName,
birthday, gender])

root.destroy() #Close the signup window after clicking signup
mainWindow.deiconify() #Bring back main menu

#Function to update gender display with an upside-down arrow
def update_gender_display(event=None):
    selected_gender = genderEntry.get()
    if selected_gender and selected_gender not in ["▼ Male", "▼ Female",
"▼ Other"]:
        genderEntry.set(selected_gender)

#Function to remove arrow when clicked
def remove_arrow(event=None):
    selected_gender = genderEntry.get().replace("▼ ", "")
    genderEntry.set(selected_gender)

#Style for rounded corners
style = ttk.Style(root)
style.theme_use('clam')

```



```

        style.configure("RoundedFrame.TFrame", background='white', borderwidth=0,
relief='flat')
        style.configure("RoundedLabel.TLabel", background='white',
foreground='#4B0082', font=("Arial", 12))
        style.configure("RoundedEntry.TEntry", fieldbackground='#f0e6fa',
foreground='#4B0082', font=("Arial", 12), borderwidth=0,
highlightthickness=0)
        style.configure("RoundedButton.TButton", background='#9370DB',
foreground='white', font=("Arial", 12, 'bold'), borderwidth=0, relief='flat')

#Function to create rounded frame
def rounded_frame(parent, **kwargs):
    frame = ttk.Frame(parent, style="RoundedFrame.TFrame", **kwargs)
    return frame

#Centering the frame (adjusted width)
floatingFrame = rounded_frame(root, padding=30)
floatingFrame.place(relx=0.5, rely=0.5, anchor="center", width=550,
height=450) #adjusted width

#Frame to hold the form
formFrame = rounded_frame(floatingFrame, padding=20)
formFrame.pack(fill="both", expand=True)

# title Label
ttk.Label(formFrame, text="Sign Up", style="RoundedLabel.TLabel",
font=("Roboto", 18, 'bold')).grid(row=0, column=0, columnspan=2, pady=10)

#Form Labels and Entries
labels = ["First Name:", "Middle Name:", "Last Name:", "Birthday
(MM/DD/YYYY):", "Gender:"]

for i, label in enumerate(labels):
    ttk.Label(formFrame, text=label,
style="RoundedLabel.TLabel").grid(row=i + 1, column=0, sticky="w", pady=7,
padx=10) #Adjusted padx

#Entry fields (adjusted width)
entry_style = {"font": "Roboto 12", "bg": "#f0e6fa", "fg": "#4B0082",
"relief": "flat", "highlightthickness": 0, "width": 25} #adjusted entry width
firstNameEntry = tk.Entry(formFrame, **entry_style)
middleNameEntry = tk.Entry(formFrame, **entry_style)
lastNameEntry = tk.Entry(formFrame, **entry_style)
birthdayEntry = tk.Entry(formFrame, **entry_style)
genderEntry = ttk.Combobox(formFrame, values=["Male", "Female", "Other"],
state="readonly", style="RoundedEntry.TEntry", width=25) #adjusted combobox
width

```

```

        entries = [firstNameEntry, middleNameEntry, lastNameEntry, birthdayEntry,
genderEntry]

        for i, entry in enumerate(entries):
            entry.grid(row=i + 1, column=1, sticky="ew", padx=10, pady=10)
#adjusted padx

        #Set default arrow display
genderEntry.set("Select Gender")

        #Bind events to modify arrow behavior
genderEntry.bind("<<ComboboxSelected>>", update_gender_display) # Add
arrow back after selection
genderEntry.bind("<Button-1>", remove_arrow) # Remove arrow when clicked

        entries = [firstNameEntry, middleNameEntry, lastNameEntry, birthdayEntry,
genderEntry]
        for i, entry in enumerate(entries):
            entry.grid(row=i + 1, column=1, sticky="ew", padx=10, pady=5)
#adjusted padx

        #Submit Button

        ttk.Button(formFrame, text="                Sign Up                ",
command=signUp, style="RoundedButton.TButton").grid(row=6, column=0,
columnspan=2, pady=15, sticky="")

        #Adjust column weights for responsiveness
formFrame.columnconfigure(0, weight=2) #increased first column weight
formFrame.columnconfigure(1, weight=3)

        mainWindow.mainloop()

```

#searchRecord.py

```

import tkinter as tk #gui
from tkinter import ttk, messagebox
from fileHandler import loadRecords #csv file
from viewRecord import viewRecordWindow

RECORDS_FILE = "records.csv"

def searchRecordWindow(mainWindow):
    #Creates the search record window.
    root = tk.Toplevel(mainWindow)
    root.title("Search A Record")
    root.geometry("700x500")

```

```

root.configure(bg='#e6e6fa')

def searchByName():
    #Search for a record and display results.
    searchCategory = searchType.get()
    searchValue = searchEntry.get().strip().lower()

    if not searchValue:
        messagebox.showerror("Error", "Search value cannot be empty.")
        root.destroy() #Close window on error
        return

    records = loadRecords(RECORDS_FILE)
    if not records:
        messagebox.showinfo("No Records", "No records found in the
system.")
        root.destroy() #Close window on error
        return

    if searchCategory == "First Name":
        results = [
            record for record in records
            if record.get("First Name",
"".strip().lower().startswith(searchValue)
        ]
    elif searchCategory == "Middle Name":
        results = [
            record for record in records
            if record.get("Middle Name",
"".strip().lower().startswith(searchValue)
        ]
    elif searchCategory == "Last Name":
        results = [
            record for record in records
            if record.get("Last Name",
"".strip().lower().startswith(searchValue)
        ]

    tree.delete(*tree.get_children())
    if results:
        for record in results:
            tree.insert("", "end", values=(record["First Name"],
record["Middle Name"], record["Last Name"], record["Birthday"],
record["Gender"]))
    else:
        messagebox.showinfo("No Record Found", "No matching record
found.")
        root.destroy() #Close window on error

```

```

def openViewRecords():
    """Open the view all records window."""
    viewRecordWindow(mainWindow)

def goBack():
    """Return to the main menu."""
    root.destroy()  #Close search window

#Title Label
tk.Label(root, text="Records Viewer", font=("Roboto", 18, 'bold'),
bg='#e6e6fa', fg='#4B0082').pack(pady=10)

#Search Frame
searchFrame = tk.Frame(root, bg='#e6e6fa')
searchFrame.pack(pady=5)

tk.Label(searchFrame, text="Search A Record:", font=("Roboto", 12),
bg='#e6e6fa').pack(side=tk.LEFT, padx=5)

#Dropdown to select search type (First Name or Last Name)
searchType = ttk.Combobox(searchFrame, values=["First Name", "Middle
Name", "Last Name"], font=("Roboto", 12), state="readonly")
searchType.set("Last Name")  # Default to Last Name
searchType.pack(side=tk.LEFT, padx=5)

searchEntry = tk.Entry(searchFrame, font=("Roboto", 12), width=30)
searchEntry.pack(side=tk.LEFT, padx=5)

tk.Button(searchFrame, text="Search", font=("Roboto", 12, 'bold'),
bg='#9370DB', fg='white', command=searchByName).pack(side=tk.LEFT, padx=5)

#Treeview Frame
treeFrame = tk.Frame(root)
treeFrame.pack(pady=10, fill=tk.BOTH, expand=True)

columns = ("First Name", "Middle Name", "Last Name", "Birthday",
"Gender")
tree = ttk.Treeview(treeFrame, columns=columns, show='headings')

for col in columns:
    tree.heading(col, text=col)
    tree.column(col, anchor=tk.CENTER, width=120)

tree.pack(fill=tk.BOTH, expand=True)

#Buttons Frame
buttonFrame = tk.Frame(root, bg='#e6e6fa')
buttonFrame.pack(pady=10)

```

```

        tk.Button(buttonFrame, text="View All Records", font=("Roboto", 12,
'bold'), bg='#9370DB', fg='white',
command=openViewRecords).pack(side=tk.LEFT, padx=10)
        tk.Button(buttonFrame, text="Back", font=("Roboto", 12, 'bold'),
bg='#8B0000', fg='white', command=goBack).pack(side=tk.LEFT, padx=10)

```

#viewRecord.py

```

import tkinter as tk #gui
from tkinter import ttk, messagebox
import csv #file handling/data persistence

RECORDS_FILE = "records.csv"

def loadRecords():
    # Load records from a CSV file and skip the first row (header)
    try:
        with open(RECORDS_FILE, mode='r', newline='') as file:
            reader = csv.reader(file)
            records = list(reader)
            return records[1:] if len(records) > 1 else []
    except FileNotFoundError:
        messagebox.showerror("Error", "The records file does not exist.")
    except Exception as e:
        messagebox.showerror("Unexpected Error", str(e))
    return []

def viewRecordWindow(mainWindow):
    # Open a new window to display all records sorted by Last Name
    root = tk.Toplevel(mainWindow)
    root.title("View Records")
    root.geometry("700x500")
    root.configure(bg='#e6e6fa')

    tk.Label(root, text="Records Viewer", font=("Roboto", 18, 'bold'),
bg='#e6e6fa', fg='#4B0082').pack(pady=3)

    treeFrame = tk.Frame(root, bg='#e6e6fa')
    treeFrame.pack(pady=3, fill=tk.BOTH, expand=True)

    # Adjust column order: Last Name first
    columns = ("Last Name", "First Name", "Middle Name", "Birthday",
"Gender")
    tree = ttk.Treeview(treeFrame, columns=columns, show='headings',
height=10)

```

```

for col in columns:
    tree.heading(col, text=col)
    tree.column(col, anchor=tk.CENTER, width=120)
tree.pack(fill=tk.BOTH, expand=True)

# Load and correctly reorder records
records = loadRecords()

# Sort records by Last Name
sorted_records = sorted(records, key=lambda record: record[2].lower())

for record in sorted_records:
    if len(record) == 5:
        first_name, middle_name, last_name, birthday, gender = record
        tree.insert("", "end", values=(last_name, first_name,
middle_name, birthday, gender))

# Close Button
tk.Button(root, text="Close", font=("Roboto", 12), bg="#8B0000",
fg="white", command=root.destroy).pack(side="bottom", pady=5,
anchor="center")

```

#fileHandler.py

```

import os
import csv

def saveRecords(filename, data):
    file_exists = os.path.isfile(filename) #check if file exists

    try:
        with open(filename, mode='a', newline='') as file:
            fieldnames = ["First Name", "Middle Name",
                "Last Name", "Birthday", "Gender"]
            writer = csv.DictWriter(file, fieldnames=fieldnames)

            #write the header only if the file does not exist
            if not file_exists:
                writer.writeheader()

            #Write the new record to the file
            writer.writerow({
                "First Name": data[0],
                "Middle Name": data[1],
                "Last Name": data[2],
                "Birthday": data[3],
                "Gender": data[4]
            })
    except Exception as e:

```

```

        print("Error saving record: ", e)

#FUNCTION load records
def loadRecords(filename):
    try:
        if not os.path.exists(filename):
            return []
        with open(filename, mode='r', newline='') as file:
            reader = csv.DictReader(file)
            records = list(reader)
            return records
    except Exception as e:
        print("Error loading records: ", e)
        return []

```

V. SUMMARY AND DISCUSSION

The User Registration Program is a Python-based application designed for efficient user management using a Tkinter-based graphical user interface (GUI). It allows users to sign up, view records, search specific entries, update details, and delete records while ensuring data persistence through CSV file handling. The program follows create and read principles and incorporates input validation and error handling mechanisms to maintain data integrity and usability. It utilizes key Python libraries such as Tkinter for the interface and CSV for file handling. The system is structured into multiple class-based modules, including `main.py` for program execution, `signUp.py` for user registration, `searchRecord.py` for searching functionalities, and `viewRecord.py` for displaying records. Each module consists of relevant methods such as `add_user()`, `search_by_name()`, `view_all_records()`, and `delete_record()` to perform core operations effectively.

By transitioning from a console-based system to an interactive GUI, the project enhances user experience and accessibility while ensuring organized record management through alphabetized sorting. While the system effectively addresses basic user management needs, future enhancements could include integrating authentication features or migrating from CSV storage to a more scalable database solution like SQLite or MySQL. Overall, this project demonstrates Python's capability in developing practical applications and highlights the importance of structured programming, GUI design, and data management techniques in software development.