**Student Examination Portal**

**Submitted by**

**Name of the Students: Soham Dey**

**Enrolment Number:** 12022002003027

**Section:** I

**Class Roll Number:** 08

**Stream:** ECE

**Subject:** Programming for Problem Solving with Python

**Subject Code:** IVC101

**Department:** Basic Science and Humanities

Under the supervision of

Swarnendu Ghosh

**Academic Year: 2022-26**

PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE FIRST SEMESTER



# CERTIFICATE OF RECOMMENDATION

We hereby recommend that the project prepared under our supervision by **Soham Dey,** entitled **Student Examination Portal** be accepted in partial of the requirements for the degree of partial fulfillment of the first semester.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Head of the Department Project Supervisor

Basic Sciences and Humanities IEM, Kolkata

# 1 Introduction

Python is a versatile and easy to use language often used in data manipulation. What separates Python from all other languages is its large number of use cases. For example, Pandas is a powerful and flexible open-source data analysis and manipulation tool built on top of the Python programming language. It allows you to easily load, manipulate, and analyze large datasets, and can greatly simplify your data wrangling and preparation tasks.

## 1.1 Objective

This project attempts to model a small scale database management system utilized by an academic institution. The objective of this project is to learn and demonstrate several python programming concepts including:

* Using python code from other files
* Importing and using third party modules
* Managing CSV data using pandas module
* Building a basic user interface
* Utilizing concepts of Object Oriented Programming

## 1.2 Organization of the Project

.

└── Student-Examination-Portal-1

├── \_pycache\_

├── csv\_files

│ ├── Batches.csv

│ ├── exam.csv

│ ├── department.csv │ ├── courses.csv

│ └── students.csv

├── Lib

├── outputs

├── pictures\_1

│ ├── reset\_btn.jpg

│ └── search\_icon\_2.jpg

├── Scripts

├── .gitignore

├── batches\_std.py

├── course\_std.py

├── department\_std.py

├── examination\_std.py

├── LICENSE

├── pandas\_tut.py

├── pyvenv.cfg

├── README.md

├── tab\_menu.py

└── students\_std.py

The code directory contains all the python code that is being executed at runtime**. batches\_std.py** is a module that exports functions that operate on a batch. Likewise**, course\_std.py** is a module that exports functions that operate on courses in the course database. Same for **department\_std.py,** which is a module that exports functions that operate on a department database.

**examination\_std.py** exports the Exam class that represents an examination being held by the institution for a particular course. **tab\_menu.py** is a file with executive permissions which imports all of the above and runs a simple graphical user interface of a tab menu to demonstrate the modules.

The **csv\_files** directory contains all the data in CSV format.

The **pictures\_1** directory contains all of the images required to make the interface.

# 2 Database Descriptions

Each student in the **student.csv** database has a unique ID, along with a name and a class roll number. Each student is associated with a single batch.

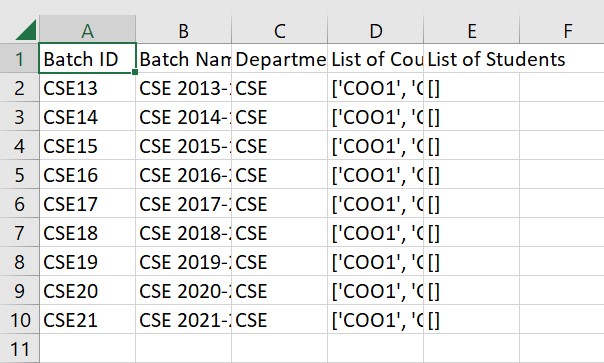
Each batch in **batch.csv** is assigned a unique ID. They also have name and a department they fall under. Each batch has a list of courses and a list of students who appear for the courses.

Each course in **course.csv** has an ID, subject name and a storage of marks obtained by each student appearing for the course.

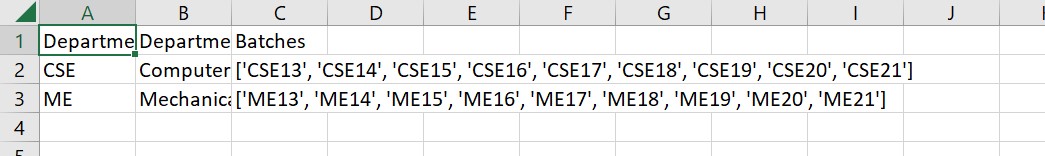
Each department in **department.csv** has an ID, name and list of batches that worked under that department.

## 2.1 Database Samples

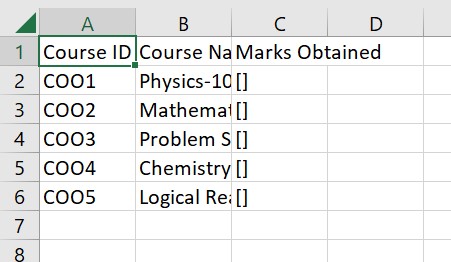
**Batches.csv**



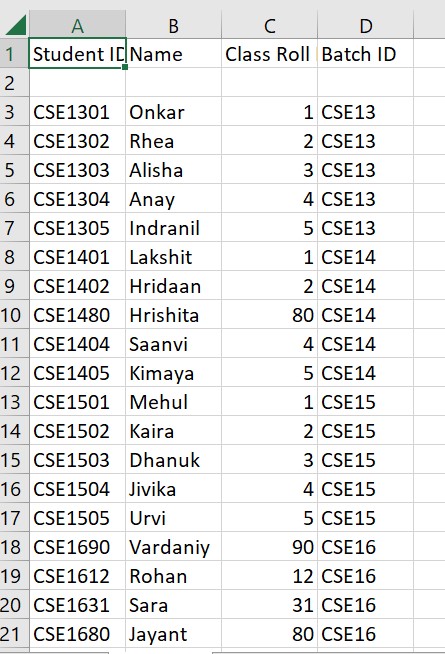
**department.csv**



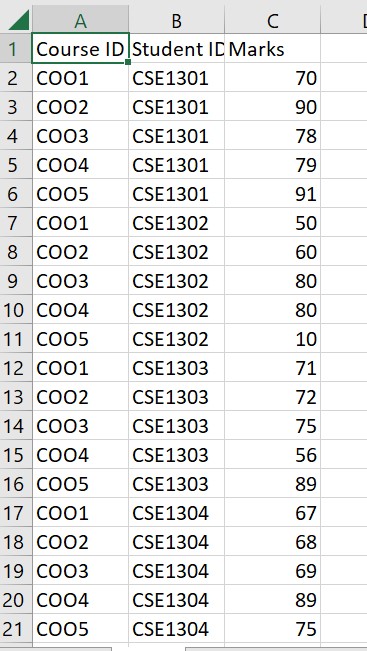
**courses.csv**



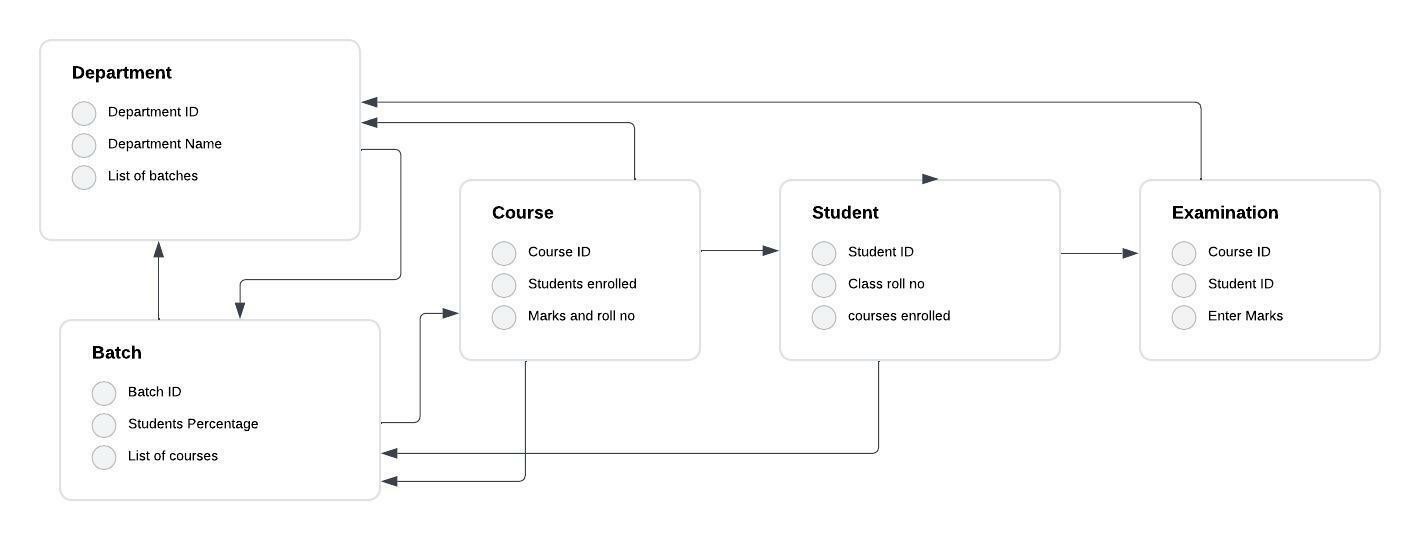
**students.csv**



**exam.csv**



# 3 Data Flow and E-R Diagrams



# 4 Programs

## Student-Examination-Portal-1/department\_std.py

**from** tkinter **import** \*

**from** tkinter **import** ttk, messagebox, Toplevel **from** PIL **import** Image, ImageTk **import** pandas **as** pd **import** csv **import** ast

**import** matplotlib.pyplot **as** plt **import** seaborn **as** plt **from** pandas\_tut **import** \*

**def** main(): root = Tk()

obj = Department(root) root.mainloop()

**class** Department: **def** \_\_init\_\_(self, root):

self.root = root

self.root.title("Department Management") screen\_width = self.root.winfo\_screenwidth() screen\_height = self.root.winfo\_screenheight() x = (screen\_width/2)-(1000/2) y = (screen\_height/2)-(500/2) self.root.geometry("1000x500+{}+{}".format(int(x)+100, int(y)))

self.root.resizable(False, False)

*# ====Menu===========*

scrollbar = Scrollbar(self.root, orient=VERTICAL) scrollbar.pack(side=RIGHT, fill=Y)

*# ========Frame1======*

self.frame1 = Frame(self.root, bg="white") self.frame1.place(x=10, y=20, height=200, width=600)

*# ====Frame2==========*

self.frame2 = Frame(self.root, bg="white")

self.frame2.place(x=620, y=20, height=200, width=350)

*# Frame3=================*

self.frame3 = Frame(self.root, bg="white", relief=RIDGE) self.frame3.place(x=10, y=230, height=250, width=960)

*# Frame1 content======*

self.depart\_id = Label(self.frame1, text="Department ID", font=(

"times new roman", 10, "bold"), bg="white").grid(row=0, column=0, pady=5, padx=5)

self.depart\_name = Label(self.frame1, text="Department Name", font=(

"times new roman", 10, "bold"), bg="white").grid(row=1, column=0, padx=5, pady=5)

self.batches\_under = Label(self.frame1, text="List Of

Batches", font=(

"times new roman", 10, "bold"), bg="white").grid(row=2, column=0, padx=5, pady=5)

*# self.scrollbar=Scrollbar(self.frame1,orient=VERTICAL)*

*# =====All variables==========* self.depart\_id\_txt\_var = StringVar() self.depart\_name\_txt\_var = StringVar() self.batches\_add\_txt\_var = StringVar()

self.txt\_search = StringVar()

self.depart\_id\_txt = Entry(self.frame1, font=( "times new roman", 15, "bold"), bg="azure2", textvariable=self.depart\_id\_txt\_var).grid(row=0, column=1, padx=5, pady=5)

self.depart\_name\_txt = Entry(self.frame1, font=( "times new roman", 15, "bold"), bg="azure2", textvariable=self.depart\_name\_txt\_var).grid(row=1, column=1, padx=5, pady=5)

self.batches\_add\_txt = Entry(self.frame1, font=( "times new roman", 15, "bold"), bg="azure2", textvariable=self.batches\_add\_txt\_var).place(x=120, y=165)

self.listbox = Listbox(self.root, bg="azure3", highlightcolor="orange", width=33, height=5)

self.listbox.place(x=129, y=100)

*# self.scrollbar.config(command=self.listbox.yview)*

*# self.scrollbar.grid(row=5 ,column=3)*

*# buttons under frame1====================*

self.btn1 = Button(self.frame1, text="Add", justify=CENTER, command=self.batches\_add).grid(row=2, column=3,)

self.btn2 = Button(self.frame1, text="Delete", justify=CENTER, command=**lambda**:

self.listbox.delete(ANCHOR)).grid(row=2, column=4,) self.btn1\_crud = Button(self.frame1, text="Add", justify=CENTER, font=(

"Comic Sans MS", 10, "bold"), command=self.crud\_add).place(x=408, y=165)

self.btn2\_crud = Button(self.frame1, text="Update", justify=CENTER, font=(

"Comic Sans MS", 10, "bold"), command=self.update\_crud).place(x=445, y=165) self.btn3\_crud = Button(self.frame1, text="Delete", justify=CENTER, font=(

"Comic Sans MS", 10, "bold"), command=self.crud\_delete).place(x=503, y=165) self.btn4\_crud = Button(self.frame1, text="Clear", justify=CENTER, font=(

"Comic Sans MS", 10, "bold"), command=self.clear\_crud).place(x=555, y=165) self

*# ===Content under frame2======================*

self.search\_image = ImageTk.PhotoImage(ImageTk.Image.open(

"pictures\_1/search\_icon\_2.jpg").resize((40, 40),

ImageTk.Image.ANTIALIAS))

self.plot\_graph\_btn = Button(self.frame2, command=**lambda**: self.create\_lineplot(self.depart\_id\_txt\_var.get()), text="Plot Graph", activeforeground="green").grid(

row=2, column=2, ipadx=20, pady=20, padx=10) self.entry\_search = Entry(self.frame2, textvariable=self.txt\_search, relief=GROOVE, bg="white", borderwidth=1, font=(

"times new roman", 17, "bold"), bd=3).grid(row=3, column=2)

self.btn\_search = Button(

self.frame2, image=self.search\_image, borderwidth=1, relief=GROOVE, bg="white", command=self.search\_data) self.btn\_search.place(x=0, y=125)

self.show\_allbtn = Button(self.frame2, text="Show All", width=10, height=2, pady=3, bg="OrangeRed3", font=( "times new roman", 10, "bold"), command=self.fetch\_data).place(x=50, y=125)

*# ==================Treeview=============================* scroll\_x = ttk.Scrollbar(self.frame3, orient=HORIZONTAL)

scroll\_y = ttk.Scrollbar(self.frame3, orient=VERTICAL) self.style = ttk.Style() self.style.theme\_use("clam")

self.style.configure("Treeview", background="grey71", foreground="black", rowheight=25, fieldbackground="grey71")

self.style.map("Treeview", background=[("selected",

"green")])

self.department\_headings = pd.read\_csv("csv\_files**\d**epartment.csv")

self.Depart\_Table = ttk.Treeview(self.frame3, columns=list( self.department\_headings.columns),

xscrollcommand=scroll\_x.set, yscrollcommand=scroll\_y.set)

scroll\_x.pack(side=BOTTOM, fill=X) scroll\_y.pack(side=RIGHT, fill=Y) scroll\_x.config(command=self.Depart\_Table.xview) scroll\_y.config(command=self.Depart\_Table.yview)

**for** i **in** self.Depart\_Table["columns"]: self.Depart\_Table.heading(i, text=i)

self.Depart\_Table["show"] = "headings" **for** i **in** self.Depart\_Table["columns"]: self.Depart\_Table.column(i, width=10) self.Depart\_Table.pack(fill=BOTH, expand=1) self.Depart\_Table["displaycolumns"] = list( self.department\_headings.columns)

self.Depart\_Table.bind("<ButtonRelease-1>", self.get\_cursor) self.fetch\_data()

self.depart\_id\_txt\_var.trace("w", self.upd)

*# ==================Toplevel\_windows=====================* **def** upd(self, \*args):

self.batches\_add\_txt\_var.set(self.depart\_id\_txt\_var.get())

**def** batches\_add(self): **if** self.batches\_add\_txt\_var.get() != "": **if** self.batches\_add\_txt\_var.get() **not** **in** self.listbox.get(0, END):

self.listbox.insert(END, self.batches\_add\_txt\_var.get())

**print**(self.listbox.get(0, END), type(self.listbox.get(0, END)))

self.batches\_add\_txt\_var.set("") **else**:

self.batches\_add\_txt\_var.set("")

**def** clear\_crud(self):

self.depart\_id\_txt\_var.set("") self.depart\_name\_txt\_var.set("") self.batches\_add\_txt\_var.set("") self.listbox.delete(0, END)

**def** crud\_delete(self):

**if** self.depart\_name\_txt\_var.get() == "" **or**

self.depart\_id\_txt\_var.get() == "" **or** list(self.listbox.get(0, END))

== []:

messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**: **try**:

excel\_filename = r"csv\_files**\d**epartment.csv" self.df = pd.read\_csv(excel\_filename)

self.df.drop(self.df.index[(

self.df[self.department\_headings.columns[0]] == self.depart\_id\_txt\_var.get())], axis=0, inplace=True)

self.df.to\_csv(excel\_filename, index=False)

self.clear\_crud() self.fetch\_data() **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**def** create\_lineplot(self, deptid):

batch\_lst = get\_lst\_of\_batches(deptid)

batch\_average = [] **for** batch **in** batch\_lst: df = pd.read\_csv("csv\_files**\e**xam.csv") dictionary = {}

sliced\_df = df[df[df.columns[1]].str[:len(batch)] == batch] crse\_lst = get\_lst\_of\_courses(batch) a = [] **for** i **in** [i[1] **for** i **in** sliced\_df.to\_numpy().tolist()]: **for** j **in** crse\_lst:

b = get\_marks(j, i)

**if** b != []:

a.append(b) **for** item **in** a: key = item[0] value = item[1] **if** key **in** dictionary:

dictionary[key].append(value) **else**:

dictionary[key] = [value] *# print(get\_average(get\_list\_of\_pct(dictionary)))*

batch\_average.append(get\_average(get\_list\_of\_pct(dictionary)))

sns.lineplot(x=batch\_lst, y=batch\_average, linestyle='--')

*# Add a title and axis labels*

plt.title('Average Percentage of All Batches') plt.xlabel('Batch List') plt.ylabel('Average Percentage')

*# Show the plot* plt.show()

**def** update\_crud(self):

**if** self.depart\_name\_txt\_var.get() == "" **or**

self.depart\_id\_txt\_var.get() == "" **or** list(self.listbox.get(0, END))

== []:

messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**: **try**:

excel\_filename = r"csv\_files**\d**epartment.csv" self.df = pd.read\_csv(excel\_filename) **for** i **in** self.df.index: **if** self.df.loc[i, self.department\_headings.columns[0]] == self.depart\_id\_txt\_var.get():

self.df.loc[i, self.department\_headings.columns[1] ] = self.depart\_name\_txt\_var.get() self.df.loc[i, self.department\_headings.columns[2]] = list( self.listbox.get(0, END)) self.df.to\_csv(excel\_filename, index=False) self.clear\_crud() self.fetch\_data()

**except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**def** fetch\_data(self):

excel\_filename = r"csv\_files**\d**epartment.csv" **if** excel\_filename: **try**:

df = pd.read\_csv(excel\_filename)

**except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root) self.Depart\_Table.delete(\*self.Depart\_Table.get\_children())

df\_rows = df.to\_numpy().tolist() **for** row **in** df\_rows:

self.Depart\_Table.insert("", END, values=row)

**def** get\_cursor(self, ev):

cursor\_row = self.Depart\_Table.focus() content = self.Depart\_Table.item(cursor\_row) self.Entry\_fill = content["values"] self.depart\_id\_txt\_var.set(self.Entry\_fill[0]) self.depart\_name\_txt\_var.set(self.Entry\_fill[1]) self.listbox.delete(0, END)

a = ast.literal\_eval(self.Entry\_fill[2]) **for** i **in** a:

self.listbox.insert(END, i)

*# for i in range(len(list(self.Entry\_fill[2]))):*

*# self.listbox.insert(END,)* **def** search\_data(self): **if** self.txt\_search.get() == "":

messagebox.showerror(

"Error", "Entry box shouldn't be empty.", parent=self.root) **else**: **try**:

df\_filtered =

self.department\_headings.loc[self.department\_headings[ self.department\_headings.columns[0]] == self.txt\_search.get()]

df\_rows = df\_filtered.to\_numpy().tolist()

self.Depart\_Table.delete(\*self.Depart\_Table.get\_children()) **for** i **in** df\_rows:

self.Depart\_Table.insert(

"", END, values=(i[0], i[1], i[2])) **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**def** crud\_add(self):

**if** self.depart\_name\_txt\_var.get() == "" **or**

self.depart\_id\_txt\_var.get() == "" **or** list(self.listbox.get(0, END))

== []:

messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**: **if** self.depart\_id\_txt\_var.get().isupper(): **try**:

excel\_filename = r"csv\_files**\d**epartment.csv" df = pd.read\_csv(excel\_filename) **if** self.depart\_id\_txt\_var.get() **not** **in** list(df.loc[:, self.department\_headings.columns[0]]): data = {self.department\_headings.columns[0]: [self.depart\_id\_txt\_var.get()], self.department\_headings.columns[1]:

[ self.depart\_name\_txt\_var.get()], self.department\_headings.columns[2]: [list(self.listbox.get(0,

END))]}

self.df = pd.DataFrame(data) self.df.to\_csv(excel\_filename, mode='a', index=False, header=False) self.clear\_crud() self.fetch\_data() **else**:

messagebox.showerror(

"Error", "Department ID should be unique.", parent=self.root) **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**else**:

messagebox.showinfo(

"Reminder", "write Department ID in Caps", parent=self.root)

**if** \_\_name\_\_ == "\_\_main\_\_": main()

## Student-Examination-Portal-1/batches\_std.py

**from** tkinter **import** \* **from** tkinter **import** ttk, messagebox **from** PIL **import** Image, ImageTk **import** pandas **as** pd **import** ast **import** threading **import** time **import** itertools **from** tabulate **import** tabulate **from** pandas\_tut **import** \* **import** matplotlib.pyplot **as** plt **import** seaborn **as** sns

**def** main(): root = Tk() obj = Batches(root) root.mainloop() **class** Batches: **def** \_\_init\_\_(self, root):

self.root = root

self.root.title("Batch Management") screen\_width = self.root.winfo\_screenwidth() screen\_height = self.root.winfo\_screenheight() x = (screen\_width/2)-(1000/2) y = (screen\_height/2)-(500/2)

self.root.geometry("1000x500+{}+{}".format(int(x)+100, int(y)))

self.root.resizable(False, False)

*# ========frames====================* self.frame1 = Frame(self.root, bg="white") self.frame1.place(x=10, y=20, height=200, width=960) self.frame2 = Frame(self.root, bg="white", relief=RIDGE) self.frame2.place(x=10, y=230, height=250, width=960) self.frame3 = Frame(self.frame1, bg="white", relief=RIDGE) self.frame3.place(x=800, y=50)

*# ===================StringVar=========================* self.combo\_id\_var = StringVar() self.entry\_name\_var = StringVar() self.course\_var = StringVar() self.txt\_search = StringVar() *# =======frame1\_content============*

self.lbl\_id = Label(self.frame1, text="Batch ID", bg="white", font=(

"times new roman", 10, "bold")).grid(row=0, column=0) self.lbl\_name = Label(self.frame1, text="Batch Name", bg="white", font=(

"times new roman", 10, "bold")).grid(row=1, column=0) self.lbl\_course\_lst = Label(self.frame1, text="List of

Courses", bg="white", font=(

"times new roman", 10, "bold")).grid(row=2, column=0)

self.combo\_id = ttk.Combobox(self.frame1, textvariable=self.combo\_id\_var, font=(

"times new roman", 15, "bold"), state="readonly", justify=RIGHT)

self.combo\_id.grid(row=0, column=1, padx=2, pady=2) self.entry\_name = Entry(self.frame1, textvariable=self.entry\_name\_var, font=(

"times new roman", 15, "bold"), bg="azure2").grid(row=1, column=1, padx=5, pady=5)

self.listbox = Listbox(self.frame1, bg="azure3", highlightcolor="orange", width=33, height=5)

self.listbox.grid(row=2, column=1) self.entry\_course = Entry(self.frame1, textvariable=self.course\_var, font=(

"times new roman", 15, "bold"), bg="azure2").grid(row=3, column=1, padx=5, pady=5)

*# ================RELOAD button================* self.reset\_image = ImageTk.PhotoImage(ImageTk.Image.open( "pictures\_1/reset\_btn.jpg").resize((40, 40), ImageTk.Image.ANTIALIAS))

self.reset\_btn = Button(self.frame1, image=self.reset\_image, height=23, borderwidth=1,

relief=GROOVE, command=self.update\_combobox()).place(x=310, y=5)

self.btn1 = Button(self.frame1, text="Add", justify=CENTER, command=self.add\_course).grid(row=2, column=3,)

self.btn2 = Button(self.frame1, text="Delete", justify=CENTER, command=**lambda**:

self.listbox.delete(ANCHOR)).grid(row=2, column=4,) self.text\_std = Text(self.frame1, height=10, width=50, bg="yellow")

self.text\_std.place(x=390, y=10)

self.btn\_pie = Button(self.frame1, text="Plot Piechart",

justify=CENTER,command=**lambda**:self.display\_piechart(self.combo\_id\_var

.get())).place(x=800, y=10)

self.btn1\_crud = Button(self.frame3, text="Add", font=("Comic Sans MS", 10, "bold"), activebackground="green", activeforeground="white", relief=RIDGE, bd=2, command=self.crud\_add).pack(side=TOP, fill=X) self.btn2\_crud = Button(self.frame3, text="Update", font=("Comic Sans MS", 10, "bold"), activebackground="green", activeforeground="white", relief=RIDGE, bd=2, command=self.update\_crud).pack(side=LEFT) self.btn3\_crud = Button(self.frame3, text="Delete", font=("Comic Sans MS", 10, "bold"), activebackground="green", activeforeground="white", relief=RIDGE, bd=2, command=self.crud\_delete).pack(side=RIGHT) self.btn4\_crud = Button(self.frame3, text="Clear", font=("Comic Sans MS", 10, "bold"), activebackground="green", activeforeground="white", relief=RIDGE, bd=2, command=self.clear\_crud).pack(side=BOTTOM) self.lbl\_search\_by = Label(self.frame1, text="Search By

BatchID", font=(

"Comic Sans MS", 10, "bold"), bg="white").place(x=800, y=125)

self.search\_image = ImageTk.PhotoImage(ImageTk.Image.open(

"pictures\_1/search\_icon\_2.jpg").resize((40, 40), ImageTk.Image.ANTIALIAS))

self.search\_entry = Entry(self.frame1, textvariable=self.txt\_search, font=(

"times new roman", 10, "bold"), bg="azure2").place(x=800, y=150)

self.btn\_search = Button(self.frame1, image=self.search\_image, height=23,

borderwidth=1, relief=GROOVE, bg="white", command=self.search\_data) self.btn\_search.place(x=880, y=170)

self.show\_allbtn = Button(self.frame1, command=self.fetch\_data, text="Show All", width=10,

height=0, pady=1, bg="OrangeRed3", font=("times new roman", 10, "bold")).place(x=800, y=170)

*# ===========Content Under frame2*

scroll\_x = ttk.Scrollbar(self.frame2, orient=HORIZONTAL) scroll\_y = ttk.Scrollbar(self.frame2, orient=VERTICAL) self.style = ttk.Style() self.style.theme\_use("clam")

self.style.configure("Treeview", background="grey71", foreground="black", rowheight=25, fieldbackground="grey71")

self.style.map("Treeview", background=[("selected",

"green")]) self.batches\_headings = pd.read\_csv("csv\_files**\B**atches.csv") self.Batch\_Table = ttk.Treeview(self.frame2, columns=list( self.batches\_headings.columns), xscrollcommand=scroll\_x.set, yscrollcommand=scroll\_y.set) scroll\_x.pack(side=BOTTOM, fill=X) scroll\_y.pack(side=RIGHT, fill=Y) scroll\_x.config(command=self.Batch\_Table.xview) scroll\_y.config(command=self.Batch\_Table.yview)

**for** i **in** self.Batch\_Table["columns"]: self.Batch\_Table.heading(i, text=i)

self.Batch\_Table["show"] = "headings" **for** i **in** self.Batch\_Table["columns"]: self.Batch\_Table.column(i, width=10) self.Batch\_Table.pack(fill=BOTH, expand=1) self.Batch\_Table["displaycolumns"] = list( self.batches\_headings.columns)

self.Batch\_Table.bind("<ButtonRelease-1>", self.get\_cursor) self.update\_combobox() self.fetch\_data()

*#*

*===========================Functions=================================*

*===* **def** update\_combobox(self):

df = pd.read\_csv("csv\_files**\d**epartment.csv") content = df[df.columns[2]] batch\_id = []

**for** i **in** range(len(content.to\_numpy().tolist())):

batch\_id.append(ast.literal\_eval(content.to\_numpy().tolist()[i]))

a = [num **for** sublist **in** batch\_id **for** num **in** sublist] self.combo\_id["values"] = a

**def** add\_course(self): **if** self.course\_var.get() != "": **if** self.course\_var.get() **not** **in** self.listbox.get(0, END):

self.listbox.insert(END, self.course\_var.get())

self.course\_var.set("")

**else**:

self.course\_var.set("")

**def** fetch\_data(self):

excel\_filename = r"csv\_files**\b**atches.csv" **if** excel\_filename: **try**:

df = pd.read\_csv(excel\_filename)

**except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

self.Batch\_Table.delete(\*self.Batch\_Table.get\_children())

df\_rows = df.to\_numpy().tolist() **for** row **in** df\_rows:

self.Batch\_Table.insert("", END, values=row)

**def** crud\_delete(self):

**if** self.combo\_id\_var.get() == "" **or** self.entry\_name\_var.get() == "" **or** list(self.listbox.get(0, END)) == []:

messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**: **try**:

excel\_filename = r"csv\_files**\B**atches.csv" self.df = pd.read\_csv(excel\_filename)

self.df.drop(self.df.index[(

self.df[self.batches\_headings.columns[0]] == self.combo\_id\_var.get())], axis=0, inplace=True)

self.df.to\_csv(excel\_filename, index=False)

self.clear\_crud() self.fetch\_data() **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**def** update\_crud(self):

**if** self.combo\_id\_var.get() == "" **or** self.entry\_name\_var.get() == "" **or** list(self.listbox.get(0, END)) == []:

messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**: **try**:

excel\_filename = r"csv\_files**\B**atches.csv" self.df = pd.read\_csv(excel\_filename) **for** i **in** self.df.index:

**if** (self.df.loc[i, self.batches\_headings.columns[0]] == self.combo\_id\_var.get()): **print**("Hello world") self.df.loc[i, self.batches\_headings.columns[1]

] = self.entry\_name\_var.get() self.df.loc[i, self.batches\_headings.columns[3]] = list(

self.listbox.get(0, END))

self.df.to\_csv(excel\_filename, index=False) self.clear\_crud() self.fetch\_data() **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**def** clear\_crud(self):

self.combo\_id\_var.set("") self.entry\_name\_var.set("") self.course\_var.set("") self.listbox.delete(0, END) self.text\_std.delete("1.0", "end")

**def** get\_cursor(self, ev):

cursor\_row = self.Batch\_Table.focus() content = self.Batch\_Table.item(cursor\_row) self.Entry\_fill = content["values"] self.combo\_id\_var.set(self.Entry\_fill[0]) self.entry\_name\_var.set(self.Entry\_fill[1]) self.listbox.delete(0, END) a = ast.literal\_eval(self.Entry\_fill[3]) **for** i **in** a:

self.listbox.insert(END, i) self.text\_std.delete(1.0, "end")

self.student\_lst = self.student\_lister(self.Entry\_fill[0]) pct\_list=self.return\_lst\_ofmarks(self.student\_lst) self.student\_lst["Percentages(%)"]=pct\_list

self.text\_std.insert(END, tabulate(self.student\_lst, tablefmt="grid"))

**def** display\_piechart(self,batchid):

student\_lst = self.student\_lister(batchid) pct\_list=self.return\_lst\_ofmarks(student\_lst) student\_lst["Percentages(%)"]=pct\_list

filtered\_rows=student\_lst.loc[:,[student\_lst.columns[0],student\_lst.c olumns[2]]] studentID = [] marks = []

**for** item **in** filtered\_rows.to\_numpy().tolist():

studentID.append(item[0]) marks.append(item[1])

colors = sns.color\_palette('bright')[0:5]

*#create pie chart*

plt.pie(marks, labels = studentID, colors = colors, autopct='%.0f%%')

plt.title(f"{batchid}: Percentage of All students") plt.show() **def** search\_data(self): **if** self.txt\_search.get() == "":

messagebox.showerror(

"Error", "Entry box shouldn't be empty.", parent=self.root) **else**: **try**:

df\_filtered =

self.batches\_headings.loc[self.batches\_headings[self.batches\_headings

.columns[0]] == self.txt\_search.get(

)]

df\_rows = df\_filtered.to\_numpy().tolist()

self.Batch\_Table.delete(\*self.Batch\_Table.get\_children()) **for** i **in** df\_rows:

self.Batch\_Table.insert(

"", END, values=(i[0], i[1], i[2], i[3], i[4])) **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**def** crud\_add(self):

**if** self.combo\_id\_var.get() == "" **or** self.entry\_name\_var.get() == "" **or** list(self.listbox.get(0, END)) == []:

messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**:

**try**:

excel\_filename = r"csv\_files**\b**atches.csv" df = pd.read\_csv(excel\_filename)

**if** self.combo\_id\_var.get() **not** **in** list(df.loc[:, self.batches\_headings.columns[0]]):

batch\_ids = self.find\_students\_in\_batch( self.combo\_id\_var.get()) data = {self.batches\_headings.columns[0]: [self.combo\_id\_var.get()], self.batches\_headings.columns[1]: [ self.entry\_name\_var.get()],

self.batches\_headings.columns[2]: [self.combo\_id\_var.get()[:-2]], self.batches\_headings.columns[3]: [list(self.listbox.get(0, END))], self.batches\_headings.columns[4]: [batch\_ids]}

self.df = pd.DataFrame(data) self.df.to\_csv(excel\_filename, mode='a', index=False, header=False) self.clear\_crud() self.fetch\_data() **else**:

messagebox.showerror(

"Error", "Department ID should be unique.", parent=self.root) **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**def** find\_students\_in\_batch(self, batch\_id): df = pd.read\_csv("csv\_files**\s**tudents.csv") in\_lst = df.loc[df[df.columns[3]] == batch\_id].to\_numpy().tolist() **return** [i[0] **for** i **in** in\_lst]

**def** student\_lister(self, batch\_id):

df = pd.read\_csv("csv\_files**\s**tudents.csv")

in\_lst = df.loc[df[df.columns[3]] ==

batch\_id, [df.columns[0], df.columns[1]]]

*#pct\_lst=self.return\_lst\_ofmarks()*

*#in\_lst["Percentages(%)"]=pct\_lst*

*#in\_lst=in\_lst.assign(Percentages=self.return\_lst\_ofmarks())*  **return** in\_lst

**def** return\_lst\_ofmarks(self,c):

*# print([i for i in self.listbox.get(0,END)])*

1. = []

**for** i **in** [i[0] **for** i **in** c.to\_numpy().tolist()]: **for** j **in** self.listbox.get(0, END):

1. = self.get\_marks(j, i) **if** b != []:

a.append(b) dictionary = {}

**for** item **in** a: key = item[0] value = item[1] **if** key **in** dictionary:

dictionary[key].append(value) **else**:

dictionary[key] = [value]

**return** get\_list\_of\_pct(dictionary)

**def** get\_marks(self, course\_id, student\_id):

excel\_file = r"csv\_files/exam.csv" df = pd.read\_csv(excel\_file)

filtered\_df = df.loc[(df['Course ID'] == course\_id) & ( df["Student ID"] == student\_id), ["Student ID", "Marks"]]

**return**

list(itertools.chain(\*filtered\_df.to\_numpy().tolist()))

**if** \_\_name\_\_ == "\_\_main\_\_":

main()

## Student-Examination-Portal-1/course\_std.py

**from** tkinter **import** \* **import** pandas **as** pd

**from** tkinter **import** ttk, messagebox **from** PIL **import** Image, ImageTk **import** ast **import** itertools **from** pandas\_tut **import** \*

**def** main(): root = Tk() obj = Course(root) root.mainloop()

**class** Course: **def** \_\_init\_\_(self, root):

self.root = root

self.root.title("Course Management") screen\_width = self.root.winfo\_screenwidth() screen\_height = self.root.winfo\_screenheight() x = (screen\_width/2)-(1000/2) y = (screen\_height/2)-(500/2)

self.root.geometry("1000x500+{}+{}".format(int(x)+100, int(y)))

self.root.resizable(False, False)

self.frame1 = Frame(self.root, bg="white") self.frame1.place(x=10, y=20, height=200, width=960) self.frame2 = Frame(self.root, bg="white", relief=RIDGE) self.frame2.place(x=10, y=230, height=250, width=960) *# =======================StringVar=======================* self.lbl\_roll\_var = StringVar() self.lbl\_marks\_var = StringVar() self.entry\_name\_var = StringVar() self.search\_entry\_var = StringVar() self.combo\_id\_var = StringVar() self.combo\_std\_id\_var = StringVar()

*# ==========Content Under frame1=====================* self.lbl\_id = Label(self.frame1, text="Course ID", bg="white", font=(

"times new roman", 10, "bold")).grid(row=0, column=0) self.lbl\_name = Label(self.frame1, text="Course Name", bg="white", font=(

"times new roman", 10, "bold")).grid(row=1, column=0) self.lbl\_marks\_obtained = Label(self.frame1, text="Marks

Obtained", bg="white", font=(

"times new roman", 10, "bold")).grid(row=2, column=0) self.combo\_id = ttk.Combobox(self.frame1, textvariable=self.combo\_id\_var, font=(

"times new roman", 15, "bold"), state="readonly", justify=RIGHT)

self.combo\_id.grid(row=0, column=1, padx=2, pady=2) self.entry\_name = Entry(self.frame1, textvariable=self.entry\_name\_var, font=(

"times new roman", 15, "bold"), bg="azure2").grid(row=1, column=1, padx=5, pady=5)

self.combo\_std\_id = ttk.Combobox(self.frame1, textvariable=self.combo\_std\_id\_var, font=(

"times new roman", 10, "bold"), width=10, state="readonly", justify=RIGHT) self.combo\_std\_id.place(x=100, y=75) self.lbl\_roll = Label(self.frame1,

textvariable=self.lbl\_roll\_var, text="Roll", bg="white", font=( "times new roman", 10, "bold"), borderwidth=1, relief="solid").place(x=200, y=75) self.lbl\_marks = Label(self.frame1,

textvariable=self.lbl\_marks\_var, text="Marks", bg="white", font=( "times new roman", 10, "bold"), borderwidth=1, relief="solid").place(x=270, y=75)

*# self.entry\_course=Entry(self.frame1,font=("times new roman",15,"bold"),bg="azure2").grid(row=3,column=1,padx=5,pady=5)*

*# ===Buttons==========*

self.btn1\_crud = Button(self.frame1, text="Add", font=("Comic Sans MS", 10, "bold"), activebackground="green",

activeforeground="white", fg="black", command=self.crud\_add).place(x=20, y=150)

self.btn2\_crud = Button(self.frame1, text="Update", font=("Comic Sans MS", 10, "bold"),

activebackground="green", activeforeground="white",

fg="black",command=self.update\_crud).place(x=60, y=150)

self.btn3\_crud = Button(self.frame1, text="Delete", font=("Comic Sans MS", 10, "bold"),

activebackground="green", activeforeground="white",

fg="black",command=self.crud\_delete).place(x=120, y=150) self.btn4\_crud = Button(self.frame1, text="Clear", font=("Comic Sans MS", 10, "bold"), activebackground="green", activeforeground="white", fg="black", command=self.clear\_crud).place(x=175, y=150)

self.search\_image = ImageTk.PhotoImage(ImageTk.Image.open(

"pictures\_1/search\_icon\_2.jpg").resize((40, 40), ImageTk.Image.ANTIALIAS))

self.search\_entry = Entry(self.frame1, textvariable=self.search\_entry\_var, font=(

"times new roman", 15, "bold"), bg="azure2").place(x=350, y=150)

self.btn\_search =

Button(self.frame1,command=self.search\_data, image=self.search\_image, height=23, borderwidth=1, relief=GROOVE, bg="white")

self.btn\_search.place(x=550, y=150)

self.show\_allbtn = Button(self.frame1, text="Show All", width=10, height=0, pady=1, bg="OrangeRed3", font=(

"times new roman", 10,

"bold"),command=self.fetch\_data).place(x=590, y=150) self.histo\_btn = Button( self.frame1, text="Histogram",

justify=CENTER,command=**lambda**:create\_histogram(self.combo\_id\_var.get(

))).place(x=350, y=100)

*# ===========Content Under frame2====================* scroll\_x = ttk.Scrollbar(self.frame2, orient=HORIZONTAL) scroll\_y = ttk.Scrollbar(self.frame2, orient=VERTICAL) self.style = ttk.Style() self.style.theme\_use("clam")

self.style.configure("Treeview", background="grey71", foreground="black", rowheight=25, fieldbackground="grey71")

self.style.map("Treeview", background=[("selected",

"green")])

self.course\_headings = pd.read\_csv("csv\_files**\c**ourses.csv") self.Course\_Table = ttk.Treeview(self.frame2, columns=list( self.course\_headings.columns),

xscrollcommand=scroll\_x.set, yscrollcommand=scroll\_y.set) scroll\_x.pack(side=BOTTOM, fill=X) scroll\_y.pack(side=RIGHT, fill=Y)

scroll\_x.config(command=self.Course\_Table.xview) scroll\_y.config(command=self.Course\_Table.yview)

**for** i **in** self.Course\_Table["columns"]: self.Course\_Table.heading(i, text=i)

self.Course\_Table["show"] = "headings" **for** i **in** self.Course\_Table["columns"]: self.Course\_Table.column(i, width=10) self.Course\_Table.pack(fill=BOTH, expand=1) self.Course\_Table["displaycolumns"] = list( self.course\_headings.columns)

self.fetch\_data()

self.Course\_Table.bind("<ButtonRelease-1>",self.get\_cursor) self.root.bind("<Button-1>", self.update\_course) self.combo\_id.bind("<<ComboboxSelected>>", self.update\_ids\_marks)

self.combo\_std\_id.bind("<<ComboboxSelected>>", self.update\_ids\_marks\_2)

*#*

*=====================Functions=======================================*

*==* **def** update\_course(self, ev): df = pd.read\_csv("csv\_files**\B**atches.csv") content = df[df.columns[3]] batch\_id = []

**for** i **in** range(len(content.to\_numpy().tolist())):

batch\_id.append(ast.literal\_eval(content.to\_numpy().tolist()[i]))

a = [num **for** sublist **in** batch\_id **for** num **in** sublist] self.combo\_id["values"] = list(set(a))

**def** update\_ids\_marks(self, ev):

df = pd.read\_csv("csv\_files/exam.csv") filtered\_df = df.loc[df['Course ID'] == self.combo\_id\_var.get()] std\_id = [] **for** i **in** filtered\_df.to\_numpy().tolist():

std\_id.append(i[1])

self.combo\_std\_id["values"] = std\_id

**def** update\_ids\_marks\_2(self, ev):

df = pd.read\_csv("csv\_files/exam.csv") filtered\_df = df.loc[df['Student ID'] == self.combo\_std\_id\_var.get()]

self.lbl\_marks\_var.set(

"Marks:"+str(filtered\_df.to\_numpy().tolist()[0][2])) self.lbl\_roll\_var.set(

"Rollno:"+filtered\_df.to\_numpy().tolist()[0][1][-2:])

**def** get\_data\_indict(self, courseid): excel\_file = r"csv\_files/exam.csv" df = pd.read\_csv(excel\_file) filtered\_df = df.loc[df['Course ID'] ==

courseid, ["Student ID", "Marks"]] **return** dict(filtered\_df.to\_numpy().tolist())

**def** clear\_crud(self):

self.combo\_id\_var.set("") self.entry\_name\_var.set("") self.combo\_std\_id\_var.set("") self.lbl\_marks\_var.set("") self.lbl\_roll\_var.set("") **def** search\_data(self): **if** self.search\_entry\_var.get() == "":

messagebox.showerror(

"Error", "Entry box shouldn't be empty.", parent=self.root) **else**: **try**:

df\_filtered =

self.course\_headings.loc[self.course\_headings[self.course\_headings.co lumns[0]] == self.search\_entry\_var.get(

)]

df\_rows = df\_filtered.to\_numpy().tolist() **for** i **in** df\_rows:

i[2] = self.get\_data\_indict(i[0])

self.Course\_Table.delete(\*self.Course\_Table.get\_children()) **for** row **in** df\_rows:

self.Course\_Table.insert("", END, values=(row[0],row[1],row[2]))

**except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root) **def** fetch\_data(self):

excel\_filename = r"csv\_files/courses.csv" **if** excel\_filename: **try**:

df = pd.read\_csv(excel\_filename) **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

self.Course\_Table.delete(\*self.Course\_Table.get\_children()) df\_rows = df.to\_numpy().tolist() **for** i **in** df\_rows:

i[2] = self.get\_data\_indict(i[0]) **for** row **in** df\_rows:

self.Course\_Table.insert("", END, values=row) **def** get\_cursor(self,ev):

cursor\_row = self.Course\_Table.focus() content = self.Course\_Table.item(cursor\_row) self.Entry\_fill = content["values"] self.combo\_id\_var.set(self.Entry\_fill[0]) self.entry\_name\_var.set(self.Entry\_fill[1]) **def** crud\_delete(self):

**if** self.combo\_id\_var.get() == "" **or** self.entry\_name\_var.get() == "":

messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**: **try**:

excel\_filename = r"csv\_files/courses.csv" self.df = pd.read\_csv(excel\_filename) self.df.drop(self.df.index[(

self.df[self.course\_headings.columns[0]] == self.combo\_id\_var.get())], axis=0, inplace=True)

self.df.to\_csv(excel\_filename, index=False)

self.clear\_crud() self.fetch\_data() **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root) **def** update\_crud(self):

**if** self.combo\_id\_var.get() == "" **or** self.entry\_name\_var.get()

== "": messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**: **try**:

excel\_filename = r"csv\_files/courses.csv" self.df = pd.read\_csv(excel\_filename) **for** i **in** self.df.index:

**if** (self.df.loc[i, self.course\_headings.columns[0]] == self.combo\_id\_var.get()):

self.df.loc[i, self.course\_headings.columns[1]

] = self.entry\_name\_var.get()

self.df.to\_csv(excel\_filename, index=False) self.clear\_crud() self.fetch\_data() **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**def** crud\_add(self):

**if** self.combo\_id\_var.get() == "" **or** self.entry\_name\_var.get()

== "": messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**: **try**:

df = pd.read\_csv("csv\_files**\c**ourses.csv")

**if** self.combo\_id\_var.get() **not** **in** list(df.loc[:, self.course\_headings.columns[0]]):

data = {self.course\_headings.columns[0]: [self.combo\_id\_var.get()], self.course\_headings.columns[1]: [ self.entry\_name\_var.get()], self.course\_headings.columns[2]: [[]]}

self.df = pd.DataFrame(data)

self.df.to\_csv("csv\_files**\c**ourses.csv", mode='a', index=False, header=False)

self.clear\_crud() self.fetch\_data() **else**:

messagebox.showerror(

"Error", "Course ID Already There.", parent=self.root)

**except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**if** \_\_name\_\_ == "\_\_main\_\_": main()

## Student-Examination-Portal-1/students\_std.py

**from** tkinter **import** \* **import** pandas **as** pd

**from** tkinter **import** ttk, messagebox **from** PIL **import** Image, ImageTk **import** ast

**from** pandas\_tut **import** \*

**def** main(): root = Tk() obj = Student(root) root.mainloop()

**class** Student: **def** \_\_init\_\_(self, root):

self.root = root

self.root.title("Student Management") screen\_width = self.root.winfo\_screenwidth() screen\_height = self.root.winfo\_screenheight() x = (screen\_width/2)-(1000/2) y = (screen\_height/2)-(500/2)

self.root.geometry("1000x500+{}+{}".format(int(x)+100, int(y)))

self.root.resizable(False, False)

*# ========frames====================* self.frame1 = Frame(self.root, bg="white") self.frame1.place(x=10, y=20, height=200, width=960) self.frame2 = Frame(self.root, bg="white", relief=RIDGE) self.frame2.place(x=10, y=230, height=250, width=960) *# ===============StringVAR==============================* self.entry\_id\_var = StringVar() self.entry\_roll\_var = StringVar() self.entry\_name\_var = StringVar() self.combo\_batch\_id\_var = StringVar() self.txt\_search = StringVar() *# =====Content Under frame1============*

self.lbl\_id = Label(self.frame1, text="Student ID", bg="white", font=(

"times new roman", 10, "bold")).grid(row=0, column=0) self.lbl\_name = Label(self.frame1, text="Student Name", bg="white", font=(

"times new roman", 10, "bold")).grid(row=1, column=0) self.lbl\_roll = Label(self.frame1, text="Class Roll Number", bg="white", font=(

"times new roman", 10, "bold")).grid(row=2, column=0) self.lbl\_marks\_obtained = Label(self.frame1, text="Batch ID", bg="white", font=(

"times new roman", 10, "bold")).grid(row=3, column=0) self.entry\_id = Entry(self.frame1,

textvariable=self.entry\_id\_var, state="readonly", font=(

"times new roman", 15, "bold"), bg="azure2").grid(row=0, column=1, padx=5, pady=5)

self.entry\_name = Entry(self.frame1, textvariable=self.entry\_name\_var, font=(

"times new roman", 15, "bold"), bg="azure2").grid(row=1, column=1, padx=5, pady=5)

self.entry\_roll = Entry(self.frame1, textvariable=self.entry\_roll\_var, font=(

"times new roman", 15, "bold"), bg="azure2").grid(row=2, column=1, padx=5, pady=5)

self.combo\_batch\_id = ttk.Combobox(self.frame1, textvariable=self.combo\_batch\_id\_var, font=(

"times new roman", 15, "bold"), state="readonly", justify=RIGHT)

self.combo\_batch\_id.grid(row=3, column=1, padx=2, pady=2) self.btn1\_crud = Button(self.frame1, text="Add", font=("Comic Sans MS", 10, "bold"), activebackground="green",

activeforeground="white", fg="black", command=self.crud\_add).place(x=20, y=150)

self.btn2\_crud = Button(self.frame1, text="Update", font=("Comic Sans MS", 10, "bold"), activebackground="green", activeforeground="white", fg="black", command=self.update\_crud).place(x=60, y=150)

self.btn3\_crud = Button(self.frame1, text="Delete", font=("Comic Sans MS", 10, "bold"), activebackground="green", activeforeground="white", fg="black", command=self.crud\_delete).place(x=120, y=150)

self.btn4\_crud = Button(self.frame1, text="Clear", font=("Comic Sans MS", 10, "bold"), activebackground="green", activeforeground="white", fg="black", command=self.clear\_crud).place(x=175, y=150) self.resetbtn = Button(self.frame1, text="Reset", bg="OrangeRed3", font=(

"times new roman", 10, "bold")).place(x=220, y=150)

self.search\_image = ImageTk.PhotoImage(ImageTk.Image.open(

"pictures\_1/search\_icon\_2.jpg").resize((40, 40), ImageTk.Image.ANTIALIAS))

self.search\_entry = Entry(self.frame1, textvariable=self.txt\_search, font=(

"times new roman", 15, "bold"), bg="azure2").place(x=350, y=150)

self.btn\_search = Button(self.frame1, command=self.search\_data,

image=self.search\_image, height=23, borderwidth=1, relief=GROOVE, bg="white") self.btn\_search.place(x=550, y=150) self.show\_allbtn = Button(self.frame1, command=self.fetch\_data, text="Show All", width=10,

height=0, pady=1, bg="OrangeRed3", font=("times new roman", 10, "bold")).place(x=590, y=150)

self.generate\_btn = Button(self.frame1, text="Generate Report Text File", font=(

"times new roman", 10, "bold"), justify=CENTER, bg="red", activebackground="yellow", activeforeground="blue",command=**lambda**:

report\_card(self.entry\_id\_var.get())).place(x=350, y=100)

*# ============Treeview==================*

scroll\_x = ttk.Scrollbar(self.frame2, orient=HORIZONTAL) scroll\_y = ttk.Scrollbar(self.frame2, orient=VERTICAL) self.style = ttk.Style() self.style.theme\_use("clam")

self.style.configure("Treeview", background="grey71", foreground="black", rowheight=25, fieldbackground="grey71")

self.style.map("Treeview", background=[("selected",

"green")])

self.student\_headings = pd.read\_csv("csv\_files**\s**tudents.csv") self.Student\_Table = ttk.Treeview(self.frame2, columns=list( self.student\_headings.columns), xscrollcommand=scroll\_x.set, yscrollcommand=scroll\_y.set) scroll\_x.pack(side=BOTTOM, fill=X) scroll\_y.pack(side=RIGHT, fill=Y)

scroll\_x.config(command=self.Student\_Table.xview) scroll\_y.config(command=self.Student\_Table.yview)

**for** i **in** self.Student\_Table["columns"]: self.Student\_Table.heading(i, text=i)

self.Student\_Table["show"] = "headings" **for** i **in** self.Student\_Table["columns"]: self.Student\_Table.column(i, width=10) self.Student\_Table.pack(fill=BOTH, expand=1) self.Student\_Table["displaycolumns"] = list( self.student\_headings.columns) self.update\_combobox() self.fetch\_data()

self.Student\_Table.bind("<ButtonRelease-1>", self.get\_cursor) self.combo\_batch\_id\_var.trace("w", self.upd) self.entry\_roll\_var.trace("w", self.upd\_2)

*# =======================Functions===================================*

**def** upd\_2(self, \*args): self.entry\_id\_var.set(

self.combo\_batch\_id\_var.get()+self.entry\_roll\_var.get())

**def** upd(self, \*args):

self.entry\_id\_var.set("")

self.entry\_id\_var.set(self.combo\_batch\_id\_var.get())

**def** update\_combobox(self):

df = pd.read\_csv("csv\_files**\d**epartment.csv") content = df[df.columns[2]] batch\_id = []

**for** i **in** range(len(content.to\_numpy().tolist())):

batch\_id.append(ast.literal\_eval(content.to\_numpy().tolist()[i]))

a = [num **for** sublist **in** batch\_id **for** num **in** sublist] self.combo\_batch\_id["values"] = a

**def** clear\_crud(self):

self.entry\_id\_var.set("") self.entry\_name\_var.set("") self.entry\_roll\_var.set("") self.combo\_batch\_id\_var.set("")

**def** crud\_delete(self):

**if** self.entry\_id\_var.get() == "" **or** self.entry\_name\_var.get() == "" **or** self.entry\_roll\_var.get() == "":

messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**: **try**:

excel\_filename = r"csv\_files**\s**tudents.csv" self.df = pd.read\_csv(excel\_filename)

self.df.drop(self.df.index[(

self.df[self.student\_headings.columns[0]] == self.entry\_id\_var.get())], axis=0, inplace=True)

self.df.to\_csv(excel\_filename, index=False) self.clear\_crud() self.fetch\_data() **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**def** update\_crud(self):

**if** self.entry\_id\_var.get() == "" **or** self.entry\_name\_var.get() == "" **or** self.entry\_roll\_var.get() == "":

messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**: **try**:

excel\_filename = r"csv\_files**\s**tudents.csv" self.df = pd.read\_csv(excel\_filename) **for** i **in** self.df.index:

**if** (self.df.loc[i, self.student\_headings.columns[0]] == self.entry\_id\_var.get()): **print**("Hello world") self.df.loc[i, self.student\_headings.columns[1]

] = self.entry\_name\_var.get()

self.df.to\_csv(excel\_filename, index=False) self.clear\_crud() self.fetch\_data()

**except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**def** fetch\_data(self):

excel\_filename = r"csv\_files**\s**tudents.csv" **if** excel\_filename: **try**:

df = pd.read\_csv(excel\_filename)

**except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

self.Student\_Table.delete(\*self.Student\_Table.get\_children())

df\_rows = df.to\_numpy().tolist() **for** row **in** df\_rows:

self.Student\_Table.insert("", END, values=row)

**def** get\_cursor(self, ev):

cursor\_row = self.Student\_Table.focus()

content = self.Student\_Table.item(cursor\_row)

self.Entry\_fill = content["values"] self.entry\_id\_var.set(self.Entry\_fill[0]) self.entry\_name\_var.set(self.Entry\_fill[1]) self.combo\_batch\_id\_var.set(self.Entry\_fill[3]) self.entry\_roll\_var.set(self.Entry\_fill[2])

**def** search\_data(self): **if** self.txt\_search.get() == "":

messagebox.showerror(

"Error", "Entry box shouldn't be empty.", parent=self.root) **else**: **try**:

df\_filtered =

self.student\_headings.loc[self.student\_headings[self.student\_headings

.columns[3]] == self.txt\_search.get(

)]

df\_rows = df\_filtered.to\_numpy().tolist()

self.Student\_Table.delete(\*self.Student\_Table.get\_children()) **for** i **in** df\_rows:

self.Student\_Table.insert(

"", END, values=(i[0], i[1], i[2],i[3])) **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**def** crud\_add(self):

**if** self.entry\_id\_var.get() == "" **or** self.entry\_name\_var.get() == "" **or** self.entry\_roll\_var.get() == "":

messagebox.showerror(

"Error", "Entry bars should not be empty", parent=self.root) **else**:

**try**:

excel\_filename = r"csv\_files**\s**tudents.csv" df = pd.read\_csv(excel\_filename)

**if** self.entry\_id\_var.get() **not** **in** list(df.loc[:, self.student\_headings.columns[0]]): data = {self.student\_headings.columns[0]: [self.entry\_id\_var.get()], self.student\_headings.columns[1]: [ self.entry\_name\_var.get()], self.student\_headings.columns[2]: [self.entry\_roll\_var.get()], self.student\_headings.columns[3]: [self.combo\_batch\_id\_var.get()]}

self.df = pd.DataFrame(data) self.df.to\_csv(excel\_filename, mode='a', index=False, header=False) self.clear\_crud()

self.fetch\_data() **else**:

messagebox.showerror(

"Error", "Student ID should be unique.", parent=self.root) **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**if** \_\_name\_\_ == "\_\_main\_\_":

main()

## Student-Examination-Portal-1/examinations\_std.py

**from** tkinter **import** \* **from** tkinter **import** ttk, messagebox **from** PIL **import** ImageTk, Image **import** pandas **as** pd **import** ast,itertools **from** pandas\_tut **import** \*

**def** main(): root = Tk() obj = Exam(root) root.mainloop()

**class** Exam: **def** \_\_init\_\_(self, root):

self.root = root

self.root.title("Examinations Management") screen\_width = self.root.winfo\_screenwidth() screen\_height = self.root.winfo\_screenheight() x = (screen\_width/2)-(1000/2) y = (screen\_height/2)-(500/2)

self.root.geometry("1000x500+{}+{}".format(int(x)+100, int(y)))

self.root.resizable(False, False)

*# ===============frames============================* self.frame1 = Frame(self.root, bg="white") self.frame1.place(x=10, y=20, height=200, width=960) self.frame2 = Frame(self.root, bg="white", relief=RIDGE) self.frame2.place(x=10, y=230, height=250, width=960)

*#==================StringVar* self.combo\_id\_var=StringVar() self.course\_id\_combo\_var=StringVar() self.entry\_roll\_var=StringVar() self.entry\_marks\_var=StringVar() self.txt\_search=StringVar()

*# ============Content\_under\_frame1==================* self.lbl\_id = Label(self.frame1, text="Student ID",

bg="white", font=(

"times new roman", 10, "bold")).grid(row=0, column=0) self.lbl\_roll = Label(self.frame1, text="Class Roll Number", bg="white", font=(

"times new roman", 10, "bold")).grid(row=1, column=0) self.lbl\_enter\_marks=Label(self.frame1, text="1.Enter

Course**\n**2.Enter Marks", bg="white", font=(

"times new roman", 10, "bold")).grid(row=2,column=0)

self.combo\_id =

ttk.Combobox(self.frame1,textvariable=self.combo\_id\_var, font=( "times new roman", 15, "bold"), state="readonly", justify=RIGHT)

self.combo\_id.grid(row=0, column=1, padx=2, pady=2) self.entry\_roll = Entry(self.frame1, font=(

"times new roman", 15,

"bold"),textvariable=self.entry\_roll\_var, bg="azure2", state="readonly").grid(row=1, column=1,padx=5,pady=5)

self.course\_id\_combo=ttk.Combobox(self.frame1,textvariable=self.cours e\_id\_combo\_var,font=("times new

roman",10,"bold"),width=10,state="readonly",justify=RIGHT) self.course\_id\_combo.place(x=120,y=75) self.marks\_entry=Entry(self.frame1, textvariable=self.entry\_marks\_var,validate="key",

validatecommand=(root.register(self.validate), '%P'),font=("times new roman", 15, "bold"),width=10, bg="azure2").place(x=215,y=75) *#===============Buttons-CRUD===================* self.btn1\_crud=Button(self.frame1,text="Add",font=("Comic

Sans MS", 10,

"bold"),activebackground="green",activeforeground="white",fg="black", command=self.crud\_add).place(x=20,y=150)

self.btn2\_crud=Button(self.frame1,text="Update",font=("Comic Sans MS", 10,

"bold"),activebackground="green",activeforeground="white",fg="black", command=self.update\_crud).place(x=60,y=150)

self.btn3\_crud=Button(self.frame1,text="Delete",font=("Comic

Sans MS", 10,

"bold"),activebackground="green",activeforeground="white",fg="black", command=self.crud\_delete).place(x=120,y=150)

self.btn4\_crud=Button(self.frame1,text="Clear",font=("Comic

Sans MS", 10,

"bold"),activebackground="green",activeforeground="white",fg="black", command=self.clear\_crud).place(x=175,y=150)

self.search\_image =

ImageTk.PhotoImage(ImageTk.Image.open("pictures\_1/search\_icon\_2.jpg")

.resize((40, 40), ImageTk.Image.ANTIALIAS))

self.reset\_image=

ImageTk.PhotoImage(ImageTk.Image.open("pictures\_1/reset\_btn.jpg").res ize((40, 40), ImageTk.Image.ANTIALIAS))

self.search\_entry=Entry(self.frame1,textvariable=self.txt\_search,font

=("times new roman",15,"bold"),bg="azure2").place(x=350,y=150) self.btn\_search=Button(self.frame1,command=self.search\_data,image=sel

f.search\_image,height=23,borderwidth=1,relief=GROOVE,bg="white") self.btn\_search.place(x=550,y=150) self.reset\_btn = Button(self.frame1, command=self.fetch\_data,image=self.reset\_image, height=23,borderwidth=1,relief=GROOVE).place(x=590, y=150) self.plot\_scatter\_btn=Button(self.frame1,command=**lambda**: create\_scatter\_plot(r"csv\_files/exam.csv"),text="Plot Scatter

Plot",font=("times new roman", 10,

"bold"),justify=CENTER,bg="red",activebackground="yellow",activeforeg round="blue").place(x=350,y=100)

*#======Treeview==============================* scroll\_x=ttk.Scrollbar(self.frame2,orient=HORIZONTAL) scroll\_y = ttk.Scrollbar(self.frame2, orient=VERTICAL) self.style=ttk.Style() self.style.theme\_use("clam")

self.style.configure("Treeview",background="grey71",foreground="black

",rowheight=25,fieldbackground="grey71")

self.style.map("Treeview",background=[("selected","green")]) self.exam\_headings=pd.read\_csv("csv\_files**\e**xam.csv")

self.exams\_Table=ttk.Treeview(self.frame2,columns=list(self.exam\_head ings.columns),xscrollcommand=scroll\_x.set,yscrollcommand=scroll\_y.set

) scroll\_x.pack(side=BOTTOM,fill=X) scroll\_y.pack(side=RIGHT,fill=Y)

scroll\_x.config(command=self.exams\_Table.xview) scroll\_y.config(command=self.exams\_Table.yview)

**for** i **in** self.exams\_Table["columns"]:

self.exams\_Table.heading(i,text=i)

self.exams\_Table["show"]="headings" **for** i **in** self.exams\_Table["columns"]: self.exams\_Table.column(i,width=10) self.exams\_Table.pack(fill=BOTH,expand=1)

self.exams\_Table["displaycolumns"]=list(self.exam\_headings.columns) self.fetch\_data()

self.exams\_Table.bind("<ButtonRelease-1>",self.get\_cursor) self.root.bind("<Button-1>",self.update\_student\_ids)

self.combo\_id.bind("<<ComboboxSelected>>",self.get\_course\_contents) self.combo\_id\_var.trace("w",self.upd)

*#====================================================================* **def** upd(self,\*args):

self.entry\_roll\_var.set("")

self.entry\_roll\_var.set(self.combo\_id\_var.get()[-2:]) **def** validate(self,P): **if** P.isdigit() **or** P=="":

**return** True

**return** False

**def** update\_student\_ids(self,ev): df=pd.read\_csv("csv\_files**\s**tudents.csv") content=df[df.columns[0]] batch\_id=[]

self.combo\_id["values"]=content.to\_numpy().tolist() **def** get\_course\_contents(self,ev): selection=self.combo\_id\_var.get() df=pd.read\_csv("csv\_files**\B**atches.csv")

df\_filtered=df.loc[df[df.columns[0]]==self.combo\_id\_var.get()[:-2]] df\_rows=df\_filtered.to\_numpy().tolist()

self.course\_id\_combo["values"]=ast.literal\_eval(df\_rows[0][3]) **def** clear\_crud(self):

self.entry\_marks\_var.set("") self.combo\_id\_var.set("") self.course\_id\_combo.set("") self.entry\_roll\_var.set("") **def** get\_cursor(self,ev):

cursor\_row = self.exams\_Table.focus() content = self.exams\_Table.item(cursor\_row) self.Entry\_fill = content["values"] self.course\_id\_combo\_var.set(self.Entry\_fill[0]) self.combo\_id\_var.set(self.Entry\_fill[1]) self.entry\_marks\_var.set(self.Entry\_fill[2]) **def** update\_crud(self):

**if** self.entry\_marks\_var.get()=="" **or** self.combo\_id\_var.get()=="" **or** self.course\_id\_combo\_var.get()=="": messagebox.showerror("Error","Entry bars should not be empty",parent=self.root) **else**: **try**:

excel\_filename = r"csv\_files/exam.csv" self.df = pd.read\_csv(excel\_filename) **for** i **in** self.df.index:

**if** (self.df.loc[i, self.exam\_headings.columns[1]]

== self.combo\_id\_var.get()):

**print**("Hello world")

self.df.loc[i, self.exam\_headings.columns[2]

] = self.entry\_marks\_var.get()

self.df.to\_csv(excel\_filename, index=False) self.clear\_crud() self.fetch\_data() **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root) **def** search\_data(self): **if** self.txt\_search.get() == "":

messagebox.showerror(

"Error", "Entry box shouldn't be empty.", parent=self.root) **else**: **try**:

df\_filtered =

self.exam\_headings.loc[self.exam\_headings[self.exam\_headings.columns[

0]] == self.txt\_search.get(

)]

df\_rows = df\_filtered.to\_numpy().tolist()

self.exams\_Table.delete(\*self.exams\_Table.get\_children()) **for** i **in** df\_rows:

self.exams\_Table.insert(

"", END, values=(i[0], i[1], i[2])) **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root) **def** crud\_delete(self):

**if** self.entry\_marks\_var.get()=="" **or** self.combo\_id\_var.get()=="" **or** self.course\_id\_combo\_var.get()=="": messagebox.showerror("Error","Entry bars should not be empty",parent=self.root) **else**: **try**:

excel\_filename = r"csv\_files/exam.csv" self.df = pd.read\_csv(excel\_filename) self.df.drop(self.df.index[(

self.df[self.exam\_headings.columns[1]] == self.combo\_id\_var.get())], axis=0, inplace=True)

self.df.to\_csv(excel\_filename, index=False)

self.clear\_crud() self.fetch\_data() **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root) **def** fetch\_data(self):

excel\_filename = r"csv\_files/exam.csv" **if** excel\_filename:

**try**:

df = pd.read\_csv(excel\_filename)

df=df.drop\_duplicates(subset=[self.exam\_headings.columns[0],self.exam

\_headings.columns[1]])

df.to\_csv("csv\_files**\e**xam.csv",index=False) df\_2= pd.read\_csv(excel\_filename)

**except** Exception **as** es: messagebox.showerror( "Error", f"Error due to: {str(es)}", parent=self.root)

self.exams\_Table.delete(\*self.exams\_Table.get\_children())

df\_rows = df\_2.to\_numpy().tolist() **for** row **in** df\_rows:

self.exams\_Table.insert("", END, values=row) **def** crud\_add(self):

**if** self.entry\_marks\_var.get()=="" **or** self.combo\_id\_var.get()=="" **or** self.course\_id\_combo\_var.get()=="": messagebox.showerror("Error","Entry bars should not be empty",parent=self.root) **else**: **try**:

*#print(self.entry\_marks\_var.get(),type(self.entry\_marks\_var.get()))* **if** int(self.entry\_marks\_var.get())>100:

messagebox.showinfo("Info","Marks Obtained should be less than or equal to 100",parent=self.root) **else**:

df = pd.read\_csv("csv\_files**\e**xam.csv") data={self.exam\_headings.columns[0]: [self.course\_id\_combo\_var.get()], self.exam\_headings.columns[1]: [ self.combo\_id\_var.get()], self.exam\_headings.columns[2]: [self.entry\_marks\_var.get()]} self.df = pd.DataFrame(data)

self.df.to\_csv("csv\_files**\e**xam.csv", mode='a', index=False, header=False)

self.clear\_crud() self.fetch\_data() **except** Exception **as** es: messagebox.showerror(

"Error", f"Error due to: {str(es)}", parent=self.root)

**if** \_\_name\_\_ == "\_\_main\_\_": main()

## Student-Examination-Portal-1/tab\_menu.py

**from** tkinter **import** \*

**from** tkinter **import** ttk, messagebox

**def** main(): root = Tk() obj = TabMenu(root) root.mainloop()

**class** TabMenu: **def** \_\_init\_\_(self, root): self.root = root

self.root.title("Student Examination Portal") screen\_width = self.root.winfo\_screenwidth() screen\_height = self.root.winfo\_screenheight() x = (screen\_width/2)-(1200/2) y = (screen\_height/2)-(500/2)

self.root.geometry("1200x500+{}+{}".format(int(x), int(y))) self.root.resizable(False, False)

*# ==================NavBar===============================* leftMenu = Frame(self.root, bd=2, relief=RIDGE, bg="white") leftMenu.place(x=0, y=0, width=200, height=499) lbl\_menu = Label(leftMenu, text="Menu", fg="azure1", font=( "times new roman", 20), bg="#009688").pack(side=TOP, fill=X)

btn\_department = Button(leftMenu, text="Department", compound=LEFT, padx=5, anchor="w", bg="white", cursor="hand2", bd=3, font=(

"times new roman", 20, "bold"), command=self.toplevel\_department).pack(side=TOP, fill=X) btn\_batch = Button(leftMenu, text="Batch", compound=LEFT, padx=5, anchor="w", bg="white", cursor="hand2", bd=3, font=(

"times new roman", 20, "bold"), command=self.toplevel\_batch).pack(side=TOP, fill=X)

btn\_course = Button(leftMenu, text="Course", compound=LEFT, padx=5, anchor="w", bg="white", cursor="hand2", bd=3, font=(

"times new roman", 20, "bold"), command=self.toplevel\_course).pack(side=TOP, fill=X)

btn\_student = Button(leftMenu, text="Student", compound=LEFT, padx=5, anchor="w", bg="white", cursor="hand2", bd=3, font=(

"times new roman", 20, "bold"), command=self.toplevel\_student).pack(side=TOP, fill=X) btn\_exam = Button(leftMenu, text="Examination", compound=LEFT, padx=5, anchor="w", bg="white", cursor="hand2", bd=3, font=(

"times new roman", 20, "bold"), command=self.toplevel\_exam).pack(side=TOP, fill=X)

btn\_logout = Button(leftMenu, text="Logout", fg="azure1", font=(

"times new roman", 20, "bold"), bg="#FF0000", command=**lambda**: self.root.destroy()).pack(side=TOP, fill=X)

self.batch\_window = None self.student\_window = None self.course\_window = None self.exam\_window = None self.dpt\_window = None

**def** toplevel\_batch(self):

**if** self.batch\_window **is** None **or** **not** self.batch\_window.winfo\_exists():

self.batch\_window = Toplevel(self.root) **from** batches\_std **import** Batches self.batches = Batches(self.batch\_window) **else**:

self.batch\_window.focus\_force()

**def** toplevel\_student(self):

**if** self.student\_window **is** None **or** **not** self.student\_window.winfo\_exists():

self.student\_window = Toplevel(self.root) **from** students\_std **import** Student self.batches = Student(self.student\_window) **else**:

self.student\_window.focus\_force()

**def** toplevel\_exam(self):

**if** self.exam\_window **is** None **or** **not** self.exam\_window.winfo\_exists():

self.exam\_window = Toplevel(self.root) **from** examinations\_std **import** Exam self.exam\_ = Exam(self.exam\_window) **else**:

self.exam\_window.focus\_force()

**def** toplevel\_course(self):

**if** self.course\_window **is** None **or** **not** self.course\_window.winfo\_exists():

self.course\_window = Toplevel(self.root) **from** course\_std **import** Course self.courses = Course(self.course\_window) **else**:

self.course\_window.focus\_force()

**def** toplevel\_department(self):

**if** self.dpt\_window **is** None **or** **not** self.dpt\_window.winfo\_exists():

self.dpt\_window = Toplevel(self.root) **from** department\_std **import** Department self.dpt = Department(self.dpt\_window) **else**:

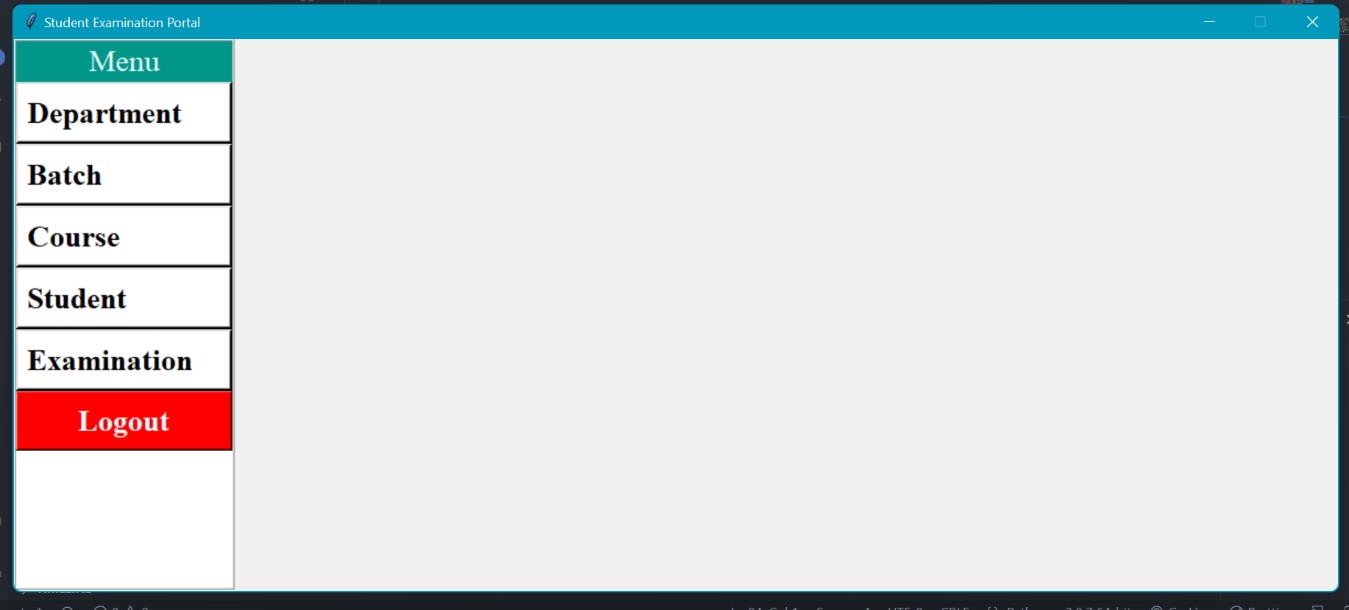
self.dpt\_window.focus\_force()

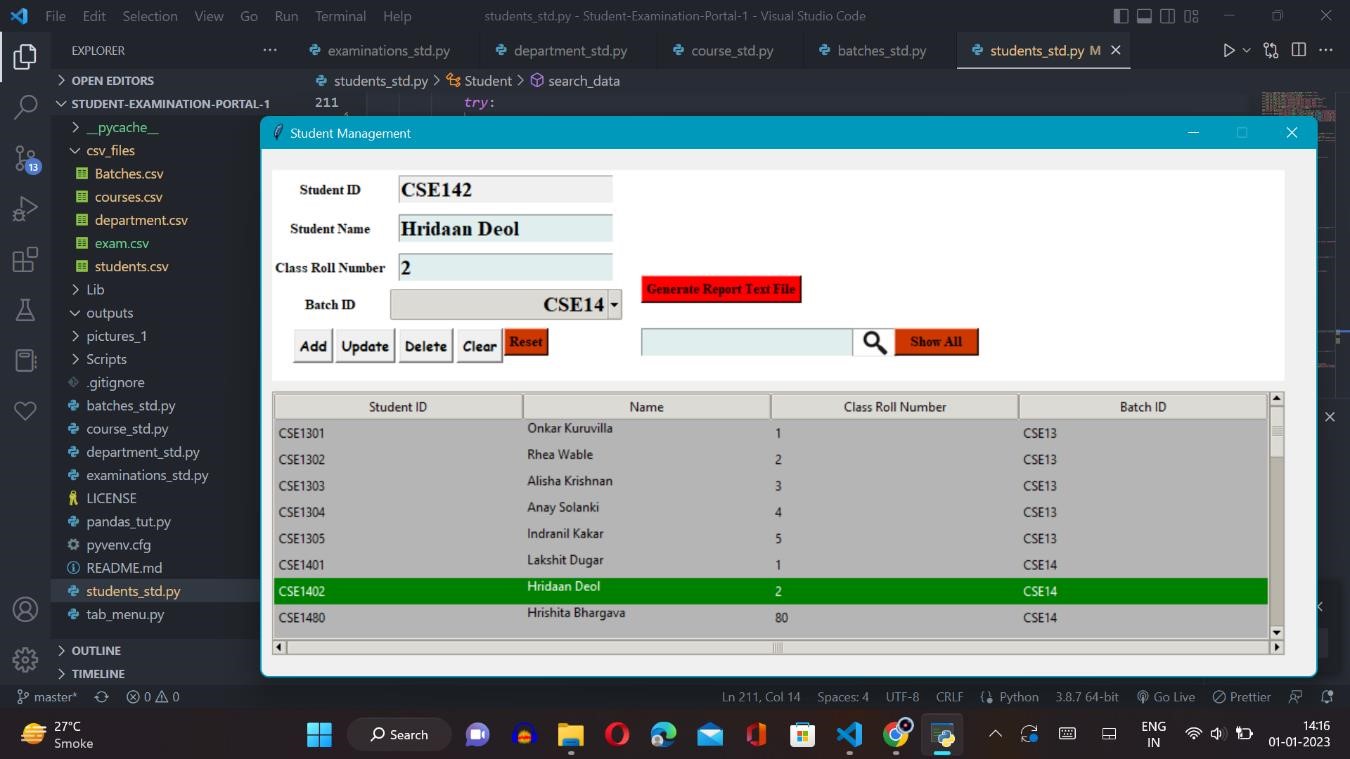
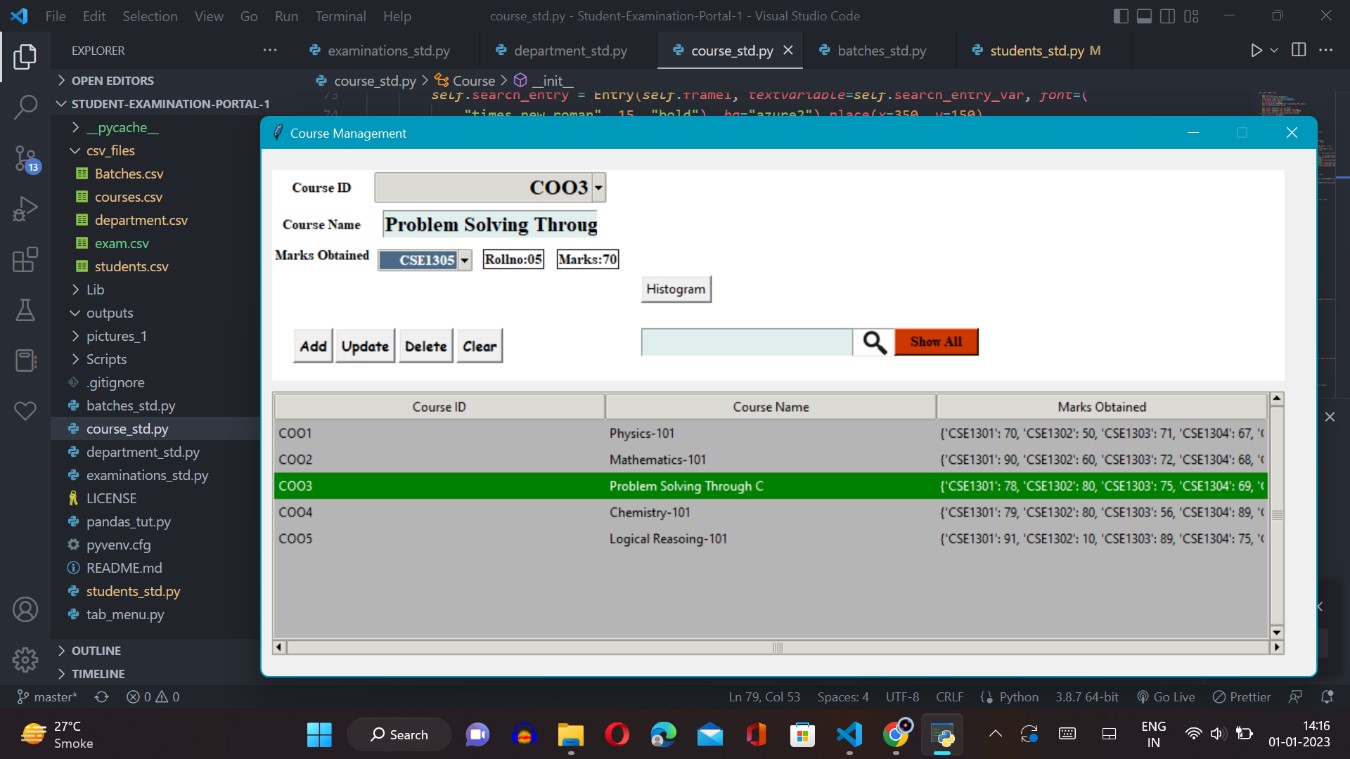
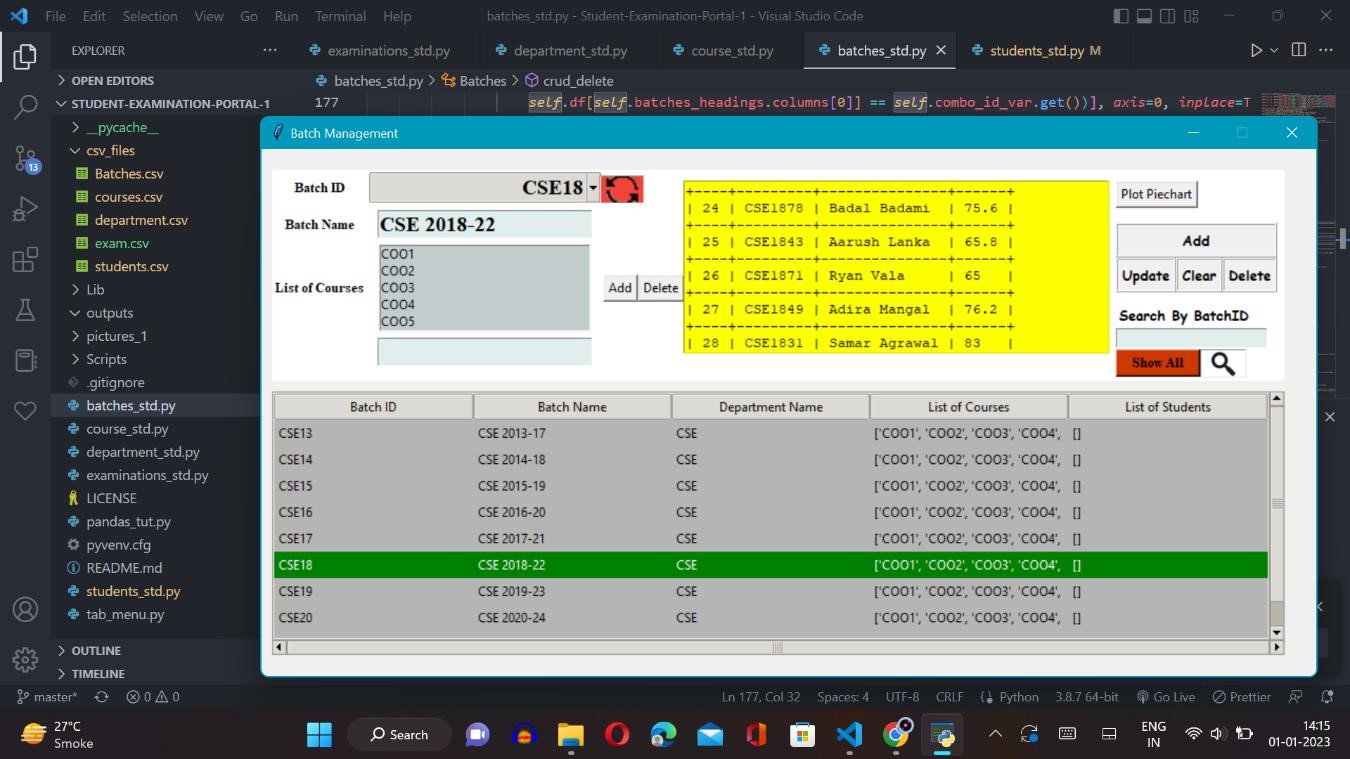
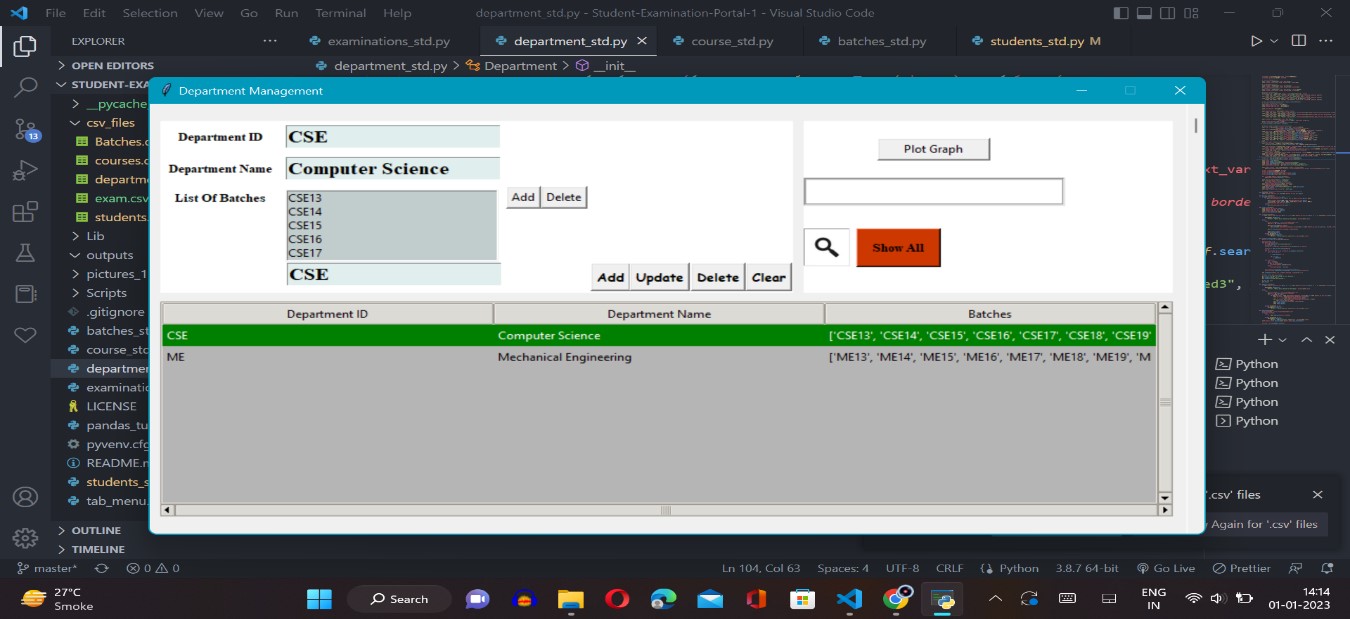
**if** \_\_name\_\_ == "\_\_main\_\_":

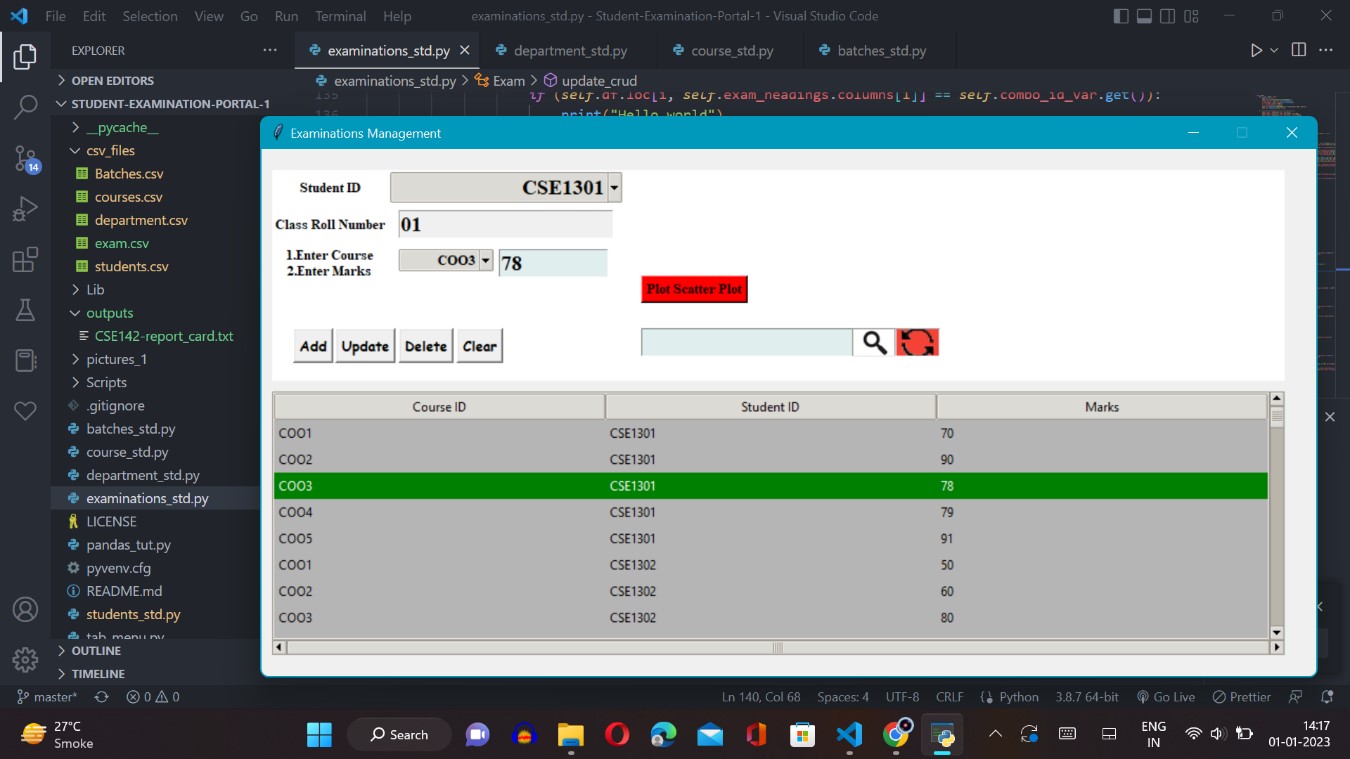
main()

# 5.Outputs

# 







**Outputs of Graphs and Piechar**

**t:**

