**PREFACE**

School Exam Center Management System project utilizing Tkinter and CSV filing. This system simplifies the process of managing exam center allocations, student data organization, and schedule management. Tkinter, a Python GUI library, provides a user-friendly interface for seamless interaction, enabling administrators to input, edit, and view exam center details with ease. CSV filing is used for efficient data storage and retrieval, ensuring portability and compatibility with various tools. This combination allows for streamlined exam center allocation, effortless data processing, and accurate record-keeping. The project aims to enhance administrative efficiency in schools, providing a reliable system for managing exam-related tasks, reducing errors, and improving overall operational productivity.

**ACKNOWLEDGEMENT**

Apart from the efforts of me, the success of any project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project.

I express deep sense of gratitude to almighty God for giving me strength for the successful completion of the project.

I express my heartfelt gratitude to my parents for constant encouragement while carrying out this project.

I gratefully acknowledge the contribution of the individuals who contributed in bringing this project up to this level, who continues to look after me despite my flaws,

I express my deep sense of gratitude to the luminary The Principal, Kanha Makhan Public School, Township (Mathura) who has been continuously motivating and extending their helping hand to us.

I am overwhelmed to express my thanks to The Administrative Officer for providing me an infrastructure and moral support while carrying out this project in the school.

My sincere thanks to TEACHER NAME, Project guide, Mentor all the above a friend, who critically reviewed my project and helped in solving each and every problem, occurred during implementation of the project

The guidance and support received from all the members who contributed and who are contributing to this project, was vital for the success of the project. I am grateful for their constant support and help.

**CERTIFICATE**

This is to certify that Ms. \_\_\_\_\_\_\_\_\_\_\_\_ Roll No: \_\_\_\_\_\_\_\_\_\_ has successfully completed the project Work entitled Project Name in the subject Computer Science (083) laid down in the regulations of CBSE for the purpose of Practical Examination in Class XII to be held in on \_\_\_\_\_\_\_\_\_\_\_\_\_.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**(Adarsh Narain)**

**Computer Science**

**Examiner:**

**Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Introduction**

The School Exam Center Management System is a software solution designed to simplify and enhance the administrative processes associated with managing exam centers in educational institutions. Developed using Python's Tkinter for the graphical user interface and CSV filing for data storage, the system offers a practical and user-friendly approach to handling exam-related tasks.

In traditional methods, the allocation of exam centers and the management of student data often involve manual processes, leading to inefficiencies, errors, and increased workload for administrators. This system addresses these challenges by automating critical tasks such as student-center assignments, record maintenance, and data retrieval.

The system allows administrators to input, edit, and view student information, including roll numbers, assigned exam centers, and schedules, all through an intuitive interface. The use of CSV files ensures lightweight, portable, and structured data storage, making it easier to share and integrate with other tools if required.

By streamlining operations, reducing errors, and improving productivity, the School Exam Center Management System aims to provide a scalable and reliable solution that caters to the needs of schools of varying sizes. Its design ensures adaptability for future enhancements, making it a valuable tool for modern educational institutions.

**Scope Of The Project**

Exam Center Allocation: Automates the assignment of students to exam centers, ensuring efficiency and equitable distribution.

Data Management: Organizes and stores student and exam data in CSV files, allowing for easy retrieval and updates.

User-Friendly Interface: Provides an intuitive GUI using Tkinter, enabling school administrators to interact with the system effortlessly.

Error Minimization: Reduces manual errors in exam center assignments and data entry by automating critical processes.

Scalability: Supports varying numbers of students and centers, making it suitable for schools of different sizes and requirements.

Improved Productivity: By streamlining operations and reducing time spent on manual tasks, the system enhances overall administrative productivity.

Secure and Reliable: Ensures data security through restricted access, maintaining the integrity and confidentiality of sensitive student information.

Student and Exam Data Management: Facilitates storing, retrieving, and modifying student details, including roll numbers, assigned centers, and schedules, in an organized manner using CSV files.

**Objective Of The system**

To streamline exam center allocation by automating the process, ensuring fairness and efficiency in assigning students to centers.

To create an intuitive interface using Tkinter for school administrators to manage student and exam data with ease.

To maintain data accuracy and reliability by utilizing CSV files for structured and error-free data storage and retrieval.

To reduce manual effort and administrative workload in managing exam-related tasks, thereby saving time and resources.

To ensure scalability and adaptability so the system can handle varying numbers of students and centers as per institutional needs.

To enable secure data handling by ensuring that only authorized personnel can access and modify sensitive exam-related information.

To facilitate easy data portability by using CSV files, allowing seamless integration with other tools or systems if needed.

To enhance productivity by providing quick and efficient tools for managing exam center data, reducing the time spent on manual tasks.

To improve data accessibility by offering centralized storage, allowing administrators to retrieve and update records instantly.

To ensure error-free operations through validation mechanisms that prevent incorrect or incomplete data entries.

To support reporting and analysis by enabling administrators to generate detailed reports on exam center allocations and student distributions.

**What is Python**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

**Advantages of Python**

* **Easy to Learn and Use**

For Beginners, Python is simple to understand and use. It's a highly developed programming language with an English-like syntax. The language is simple to adapt as a result of these factors. Because of its simplicity, Python's fundamentals can be implemented faster than those in other programming languages.

* **Free and Open-Source**

Python is distributed under an open-source license approved by the Open-Source Initiative (OSI). As a result, users can work on it and distribute it. Users can download the source code, modify it, and even distribute their Python version. Companies that wish to modify a specific behavior and build their version will benefit.

* **Rapid Development**

Users can create new kinds of applications using the Python programming language. Because of its versatility, this language permits the operator to try new things. Because of the language, the user is not prevented from trying something new. Python is favored in these scenarios since other programming languages lack the flexibility and freedom that Python does.

* **Interpreted Language**

Python is an interpreted language, implying that the code is implemented line by line. This is one of the features that makes it simple to use. In the event of an error, it halts the process and reports the problem. Python only shows one error, even if the program has multiple errors. This makes debugging easier.

* **Wide Range of Libraries and Frameworks**

Python includes a huge number of libraries that the user can use. The standard library in Python is immense, and it includes almost every function imaginable. Large and supportive communities, as well as corporate sponsorship, have contributed to this. When working with Python, users do not need to use external libraries.

* **Dynamically Typed**

Until we run the program, Python has no idea what kinds of parameter we're talking about. It allocates the data type automatically during execution. Variables and their data types do not need to be declared by the programmer.

* **Portability**

Many other languages, including C/C++, demand that user must change their code to run on different platforms. Python, on the contrary, is not equivalent to other programming languages. It only needs to be written once, and then it can be run anywhere. However, the user should avoid involving any system-dependent features.

* **Strong Community Support**

Python is a programming language generated many years ago and has a large community that can assist programmers of all experience levels, from rookies to specialists. Python's community has helped it grow quickly in comparison to other languages. The Python programming language comes with many guides, instructional videos, and highly understandable documentation to help developers learn the language faster and more effectively.

**Applications of Python**

Here are the most common applications of Python in the world of programming.

* **Visualization of data**

Another popular and growing area of interest is data visualization. Again, it plays to many of Python's strengths. Python, in addition to its flexibility and open-source nature, offers a wide range of graphic libraries with a variety of features.

* **Artificial intelligence and machine learning**

Python is a stable, versatile, easy yet effective programming language that is ideal for various machine learning (ML) and artificial intelligence (AI) projects. Thanks to the abundance of Python machine learning and artificial intelligence libraries and packages, python is undoubtedly one of the most widely used programming languages among data analysts and data scientists.

* **Data Analytics**

Python makes sense for data science and analytics. The language is simple to learn, flexible, and very well supported, making it quick and simple to use for data analysis. It's useful for attempting to manipulate data and performing repetitive tasks when working directly with huge amounts of data.

* **Desktop GUI**

Python is a much more interactive language for programmers than other languages that enable developers to quickly and easily create graphical user interfaces. It includes many built-in tools such as PyQT, wxWidgets, kivy, and many other packages and libraries that can be used to create a fully operational GUI efficiently and securely.

* **Web Development**

Web design and development using Python is very efficient. This is largely due to the innumerable Python web development paradigms, such as Django, Flask, and Piramyd that are available. These frameworks have been used by sites and services such as Spotify, Reddit, and Mozilla.

* **Audio and Visual Applications**

Python's audio and visual applications are, without a doubt, its most impressive feature. Python comes with many frameworks and functions to help us complete our tasks flawlessly. Netflix, Hubspot, and YouTube are just a few examples of Python-based applications. Libraries like Mignus, Dejavu, Pyo, SciPy, and OpenCV can help.

* **CAD Applications**

Digitally created 3D and 2D models. Architects, construction managers, and product designers use this application to build things with extremely high consistency, which has replaced manual drift. Python comes pre-installed with fantastic applications such as Blender, Open Cascade, FreeCAD, and many others that help us design products quickly. Technical drawing, dynamic system development and import are all enhanced features.

* **Scientific Computing**

Python is becoming more widely used in finance, particularly in domains such as qualitative data analysis and qualitative analysis. It can be a useful tool for predicting asset price trends, automating workflows across multiple data sources, and deciding asset price patterns and predictions.

* **Web Scraping Application**

Web scraping is a fully automated method of extracting data from webpages in a more efficient and timely manner. Researchers, organizations, and analysts use the data for a wide range of purposes. Python has a number of characteristics that make it ideal for web scraping, including:

* A simple syntax that improves readability and saves time.
* Simple to use and comprehend
* The web scraping process is simple and efficient thanks to a variety of libraries and tools such as pandas, matplotlib, and Selenium.

Other useful and important applications of Python language are:

* Game Development
* Internet of Things (IoT)
* Network Programming
* Robotics
* Natural Language Processing (NLP)

**Real-World examples for Python applications**

Python is used everywhere now a days. The real-world example tasks performed by Python applications are given below.

**YouTube -** Video transcoding, content recommendations, and analytics.

**Dropbox -** Building and maintaining the file hosting service's server infrastructure.

**NASA -** Scientific computing, data analysis, and simulation tasks.

**Spotify -** Data analysis, recommendation systems, and backend services in the music streaming platform.

**Instagram -** Backend infrastructure and data processing in the popular social media platform.

**Google -** Web crawling, data analysis, and automation.

**File Handling in Python**

File handling in Python is a powerful and versatile tool that can be used to perform a wide range of operations. However, it is important to carefully consider the advantages and disadvantages of file handling when writing Python programs, to ensure that the code is secure, reliable, and performs well.

**Python File Handling**

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but like other concepts of Python, this concept here is also easy and short. Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun. Let’s start with the reading and writing files.

**Advantages of File Handling**

* **Versatility**: File handling in Python allows you to perform a wide range of operations, such as creating, reading, writing, appending, renaming, and deleting files.
* **Flexibility**: File handling in Python is highly flexible, as it allows you to work with different file types (e.g. text files, binary files, CSV files, etc.), and to perform different operations on files (e.g. read, write, append, etc.).
* **User**–**friendly**: Python provides a user-friendly interface for file handling, making it easy to create, read, and manipulate files.
* **Cross-platform**: Python file-handling functions work across different platforms (e.g. Windows, Mac, Linux), allowing for seamless integration and compatibility.

**CSV File Reading and Writing**

The so-called CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases. CSV format was used for many years prior to attempts to describe the format in a standardized way in RFC 4180. The lack of a well-defined standard means that subtle differences often exist in the data produced and consumed by different applications. These differences can make it annoying to process CSV files from multiple sources. Still, while the delimiters and quoting characters vary, the overall format is similar enough that it is possible to write a single module which can efficiently manipulate such data, hiding the details of reading and writing the data from the programmer.

The csv module implements classes to read and write tabular data in CSV format. It allows programmers to say, “write this data in the format preferred by Excel,” or “read data from this file which was generated by Excel,” without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.

The csv module’s reader and writer objects read and write sequences. Programmers can also read and write data in dictionary form using the DictReader and DictWriter classes.

**Binary Files**

Generally, binary means two. In computer science, binary files are stored in a binary format having digits 0’s and 1’s. For example, the number 9 in binary format is represented as ‘1001’. In this way, our computer stores each and every file in a machine-readable format in a sequence of binary digits. The structure and format of binary files depend on the type of file. Image files have different structures when compared to audio files. However, decoding binary files depends on the complexity of the file format. In this article, let’s understand the reading of binary files.

**Reading Binary Files in Python**

Reading binary files is an important skill for working with data (non-textual) such as images, audio, and videos. Using file mode and the “read” method you can easily read binary files. Python has the ability to handle the data and consent provides various help with certain criteria. Whether you are dealing with multimedia files, compressed data, or custom binary formats, Python’s ability to handle binary data empowers you to create powerful and versatile applications for a wide range of use cases. In this article, you will learn What binary files are and how to read data into a byte array, and Read binary data into chunks? and so on.

**SOURCE CODE**

**import tkinter as tk**

**from tkinter import ttk, messagebox**

**import csv**

**import os**

**# CSV file name**

**CSV\_FILE = "exam\_centers.csv"**

**# Ensure the CSV file exists**

**def init\_csv():**

**if not os.path.exists(CSV\_FILE):**

**with open(CSV\_FILE, mode="w", newline="") as file:**

**writer = csv.writer(file)**

**writer.writerow(["Center Code", "Center Name", "District", "State", "Student Roll No", "Class (10/12)"])**

**# Function to read data from CSV**

**def read\_data():**

**data = []**

**if os.path.exists(CSV\_FILE):**

**with open(CSV\_FILE, mode="r") as file:**

**reader = csv.reader(file)**

**next(reader)  # Skip the header row**

**for row in reader:**

**data.append(row)**

**return data**

**# Function to write a new record to the CSV file**

**def write\_data(record):**

**with open(CSV\_FILE, mode="a", newline="") as file:**

**writer = csv.writer(file)**

**writer.writerow(record)**

**# Function to update an existing record in the CSV file**

**def update\_data(center\_code, student\_roll\_no, new\_center\_name, new\_district, new\_state, new\_class):**

**data = read\_data()**

**updated = False**

**with open(CSV\_FILE, mode="w", newline="") as file:**

**writer = csv.writer(file)**

**writer.writerow(["Center Code", "Center Name", "District", "State", "Student Roll No", "Class (10/12)"])  # Write header**

**for row in data:**

**if row[0] == center\_code and row[4] == student\_roll\_no:**

**row[1] = new\_center\_name**

**row[2] = new\_district**

**row[3] = new\_state**

**row[5] = new\_class**

**updated = True**

**writer.writerow(row)**

**return updated**

**# Function to delete a record from the CSV file**

**def delete\_data(center\_code):**

**data = read\_data()**

**with open(CSV\_FILE, mode="w", newline="") as file:**

**writer = csv.writer(file)**

**writer.writerow(["Center Code", "Center Name", "District", "State", "Student Roll No", "Class (10/12)"])  # Write header**

**for row in data:**

**if row[0] != center\_code:**

**writer.writerow(row)**

**# Admin panel**

**def admin\_panel():**

**def refresh\_table():**

**# Clear the treeview before adding new data**

**for row in tree.get\_children():**

**tree.delete(row)**

**for row in read\_data():**

**tree.insert("", "end", values=row)**

**def add\_record():**

**try:**

**start\_roll = int(entry\_start\_rollno.get())**

**end\_roll = int(entry\_end\_rollno.get())**

**class\_name = entry\_class.get()**

**center\_code = entry\_center\_code.get()**

**center\_name = entry\_center\_name.get()**

**district = entry\_district.get()**

**state = entry\_state.get()**

**if start\_roll > end\_roll:**

**messagebox.showwarning("Input Error", "Starting Roll No cannot be greater than Ending Roll No.")**

**return**

**# Generate records for all students in the range**

**for roll\_no in range(start\_roll, end\_roll + 1):**

**new\_record = (**

**center\_code, center\_name, district, state, str(roll\_no), class\_name**

**)**

**write\_data(new\_record)**

**refresh\_table()**

**messagebox.showinfo("Success", f"Records for roll numbers {start\_roll} to {end\_roll} added successfully!")**

**clear\_fields()**

**except ValueError:**

**messagebox.showwarning("Input Error", "Please enter valid numeric values for Roll No.")**

**def update\_record():**

**selected\_item = tree.selection()  # Get selected row from the treeview**

**if not selected\_item:**

**messagebox.showwarning("Selection Error", "Please select a record to update.")**

**return**

**center\_code = tree.item(selected\_item[0])['values'][0]  # Extract center\_code from the selected row**

**student\_roll\_no = tree.item(selected\_item[0])['values'][4]  # Extract student\_roll\_no from the selected row**

**# Gather the updated values from the entry fields**

**new\_center\_name = entry\_center\_name.get()**

**new\_district = entry\_district.get()**

**new\_state = entry\_state.get()**

**new\_class = entry\_class.get()**

**if messagebox.askyesno("Update Confirmation", f"Are you sure you want to update record with Center Code: {center\_code}, Roll No: {student\_roll\_no}?"):**

**if update\_data(center\_code, student\_roll\_no, new\_center\_name, new\_district, new\_state, new\_class):**

**refresh\_table()  # Refresh the table to show updated records**

**messagebox.showinfo("Success", f"Record updated successfully.")**

**else:**

**messagebox.showwarning("Update Failed", "Record not found.")**

**def delete\_record():**

**selected\_item = tree.selection()  # Get selected row from the treeview**

**if not selected\_item:**

**messagebox.showwarning("Selection Error", "Please select a record to delete.")**

**return**

**center\_code = tree.item(selected\_item[0])['values'][0]  # Extract center\_code from the selected row**

**if messagebox.askyesno("Delete Confirmation", f"Are you sure you want to delete record with Center Code: {center\_code}?"):**

**delete\_data(center\_code)  # Call the delete\_data function**

**refresh\_table()  # Refresh the table to show updated records**

**messagebox.showinfo("Success", f"Record with Center Code: {center\_code} deleted successfully.")**

**def clear\_fields():**

**entry\_center\_code.delete(0, tk.END)**

**entry\_center\_name.delete(0, tk.END)**

**entry\_district.delete(0, tk.END)**

**entry\_state.delete(0, tk.END)**

**entry\_class.delete(0, tk.END)**

**entry\_start\_rollno.delete(0, tk.END)**

**entry\_end\_rollno.delete(0, tk.END)**

**admin\_window = tk.Toplevel(root)**

**admin\_window.title("Admin Panel")**

**admin\_window.geometry("900x500")**

**# Set background color**

**admin\_window.configure(bg="#f0f8ff")**

**frame = tk.Frame(admin\_window, bg="#f0f8ff")**

**frame.pack(fill="both", expand=True)**

**# Entry fields for new data**

**labels = [**

**"Center Code", "Center Name", "District", "State",**

**"Class (10/12)", "Start Roll No", "End Roll No"**

**]**

**entries = []**

**for i, label in enumerate(labels):**

**lbl = tk.Label(frame, text=label, bg="#f0f8ff", font=("Helvetica", 10, "bold"))**

**lbl.grid(row=i, column=0, padx=5, pady=5)**

**entry = tk.Entry(frame)**

**entry.grid(row=i, column=1, padx=5, pady=5)**

**entries.append(entry)**

**(entry\_center\_code, entry\_center\_name, entry\_district, entry\_state,**

**entry\_class, entry\_start\_rollno, entry\_end\_rollno) = entries**

**# Buttons for operations**

**btn\_add = tk.Button(frame, text="Add Records", bg="#20b2aa", fg="white", font=("Helvetica", 10, "bold"), command=add\_record)**

**btn\_add.grid(row=0, column=2, padx=5, pady=5)**

**btn\_update = tk.Button(frame, text="Update Selected", bg="#ff8c00", fg="white", font=("Helvetica", 10, "bold"), command=update\_record)**

**btn\_update.grid(row=1, column=2, padx=5, pady=5)**

**btn\_delete = tk.Button(frame, text="Delete Selected", bg="#dc143c", fg="white", font=("Helvetica", 10, "bold"), command=delete\_record)**

**btn\_delete.grid(row=2, column=2, padx=5, pady=5)**

**# Treeview for displaying data (without initial headers or data)**

**tree = ttk.Treeview(admin\_window, show="headings")**

**# Note: We're not adding any initial data here and not showing the headers automatically**

**tree.pack(fill="both", expand=True)**

**# Refresh the table when the admin panel is opened**

**refresh\_table()**

**# Student panel (unchanged)**

**def student\_panel():**

**def fetch\_student\_details():**

**roll\_no = entry\_student\_rollno.get().strip()**

**class\_name = entry\_student\_class.get().strip()**

**if not roll\_no or not class\_name:**

**lbl\_center\_details.config(text="Please enter both Roll No and Class.")**

**return**

**data = read\_data()**

**result = None**

**for row in data:**

**if row[4].strip() == roll\_no and row[5].strip() == class\_name:**

**result = row**

**break**

**if result:**

**lbl\_center\_details.config(**

**text=f"Center Code: {result[0]}\n"**

**f"Center Name: {result[1]}\n"**

**f"District: {result[2]}\n"**

**f"State: {result[3]}"**

**)**

**else:**

**lbl\_center\_details.config(text="No data found for the provided Roll No and Class.")**

**student\_window = tk.Toplevel(root)**

**student\_window.title("Student Panel")**

**student\_window.geometry("800x400")**

**# Set background color**

**student\_window.configure(bg="#ffe4e1")**

**frame = tk.Frame(student\_window, bg="#ffe4e1")**

**frame.pack(pady=20)**

**# Labels and entry fields for student details**

**lbl\_roll\_no = tk.Label(frame, text="Enter Roll No:", bg="#ffe4e1", font=("Helvetica", 10, "bold"))**

**lbl\_roll\_no.grid(row=0, column=0, padx=5, pady=5)**

**entry\_student\_rollno = tk.Entry(frame)**

**entry\_student\_rollno.grid(row=0, column=1, padx=5, pady=5)**

**lbl\_class = tk.Label(frame, text="Enter Class (10/12):", bg="#ffe4e1", font=("Helvetica", 10, "bold"))**

**lbl\_class.grid(row=1, column=0, padx=5, pady=5)**

**entry\_student\_class = tk.Entry(frame)**

**entry\_student\_class.grid(row=1, column=1, padx=5, pady=5)**

**# Button to fetch details**

**btn\_fetch = tk.Button(frame, text="Fetch Center Details", bg="#4682b4", fg="white", font=("Helvetica", 10, "bold"), command=fetch\_student\_details)**

**btn\_fetch.grid(row=2, column=0, columnspan=2, pady=10)**

**# Label for showing center details**

**lbl\_center\_details = tk.Label(frame, text="Center Details will be displayed here.", bg="#ffe4e1", font=("Helvetica", 10, "bold"))**

**lbl\_center\_details.grid(row=3, column=0, columnspan=2, padx=5, pady=5)**

**# Login screen**

**root = tk.Tk()**

**root.title("Exam Center Management")**

**root.geometry("300x200")**

**# Set background color**

**root.configure(bg="#fffacd")**

**frame = tk.Frame(root, bg="#fffacd")**

**frame.pack(pady=20)**

**btn\_admin = tk.Button(frame, text="Admin Login", width=15, bg="#4682b4", fg="white", font=("Helvetica", 10, "bold"), command=admin\_panel)**

**btn\_admin.grid(row=0, column=0, padx=5, pady=10)**

**btn\_student = tk.Button(frame, text="Student Login", width=15, bg="#32cd32", fg="white", font=("Helvetica", 10, "bold"), command=student\_panel)**

**btn\_student.grid(row=1, column=0, padx=5, pady=10)**

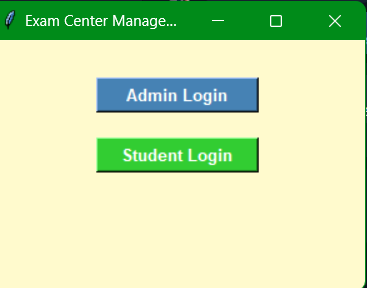
**# Initialize CSV file**

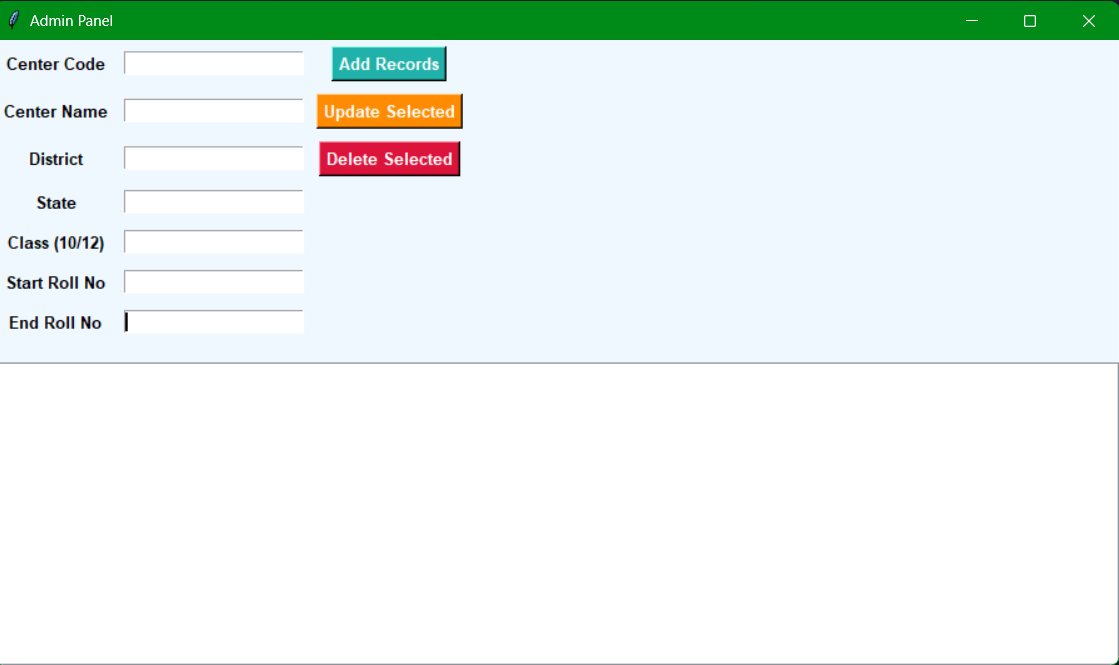
**init\_csv()**

**root.mainloop()**

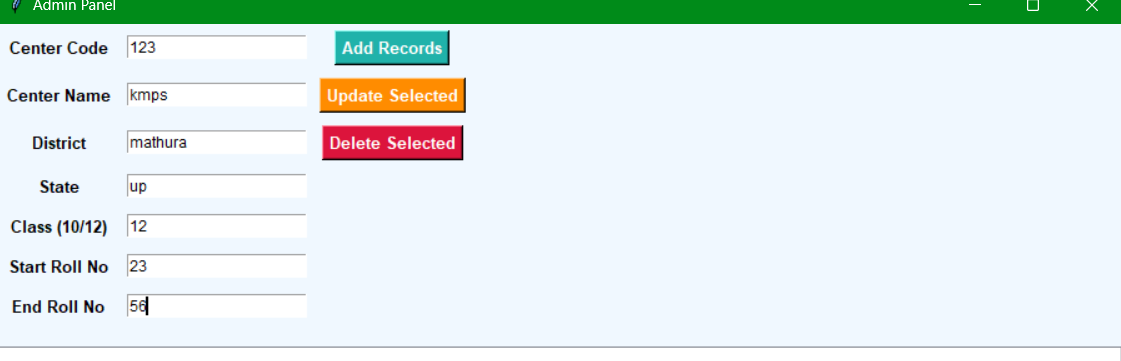
**OUTPUT**

**MAIN PANNEL**

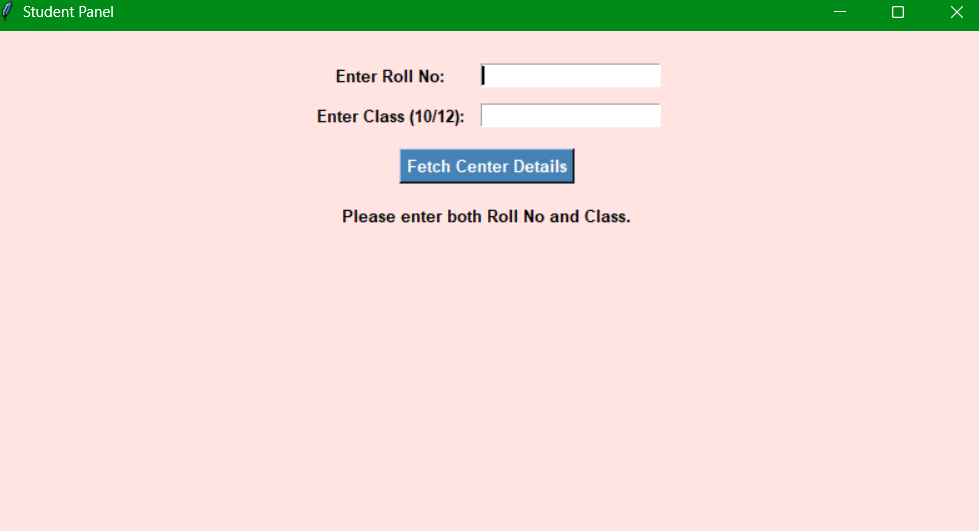
****

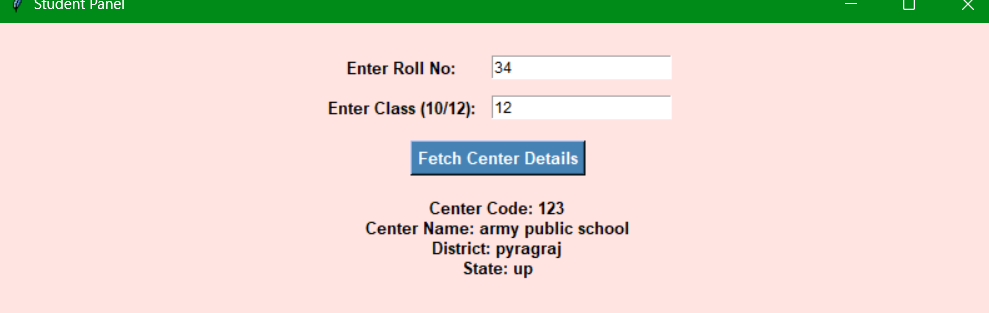
**ADMIN PANNEL  
**

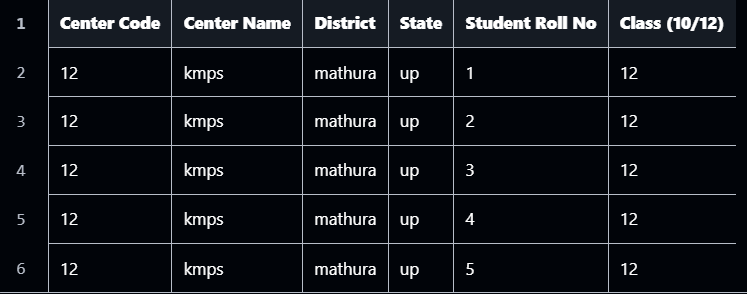
**ADDING RECORDS**

****

**STUDENT PANNEL**

****

**VIEW RECORD FOR STUDENT  
**

**DATA SAVED IN FILE(exam\_centers.csv)  
**