# STUDENT MANAGEMENT LIST SYSTEM

# USING PYTHON

**Submitted by:**

## NISHANTHINI.K

**Guided by:**

**MONISHA**

# Abstract:


- ❖ Developed a desktop application using Python and Tkinter for student data management.
- ❖ Manages both student personal details and academic marks efficiently.
- ❖ Supports mark entry for five subjects: Tamil, English, Mathematics, Science, and Social Science.
- ❖ Automatically calculates average marks, percentage, and assigns rank.
- ❖ Includes features like edit, delete, clear, select-view-only, and scroll functionality.
- ❖ Stores data using CSV files for easy access and portability.
- ❖ Designed with a user-friendly interface suitable for educational use.

## AIM:

To create a simple and effective desktop system for managing student information and marks using Python and Tkinter, with automated calculations and easy data management.

## INTRODUCTION:

- The Student Management List System is a desktop application developed using Python and Tkinter.
- It is designed to manage both personal and academic details of students efficiently.
- Users can input marks for five subjects and automatically calculate average, percentage, and rank.
- The system provides features like edit, delete, view-only select, clear, and scroll functionality.
- Data is stored using CSV files for easy access and long-term storage.
- It offers a simple and effective solution for managing student records in academic settings.

### Software & Hardware Requirements:

**Software Requirements:**

- **Programming Language:** Python (version 3.x)
- **GUI Framework:** Tkinter (comes with Python)
- **IDE/Code Editor:** VS Code, or any Python-compatible editor
- **Data Storage:** CSV file handling (built-in Python module)
- **Operating System:** Windows

**Hardware Requirements:**

- **Processor:** Minimum 1.6 GHz or higher (Dual-core or above)
- **RAM:** Minimum 2 GB (4 GB recommended)
- **Storage:** At least 100 MB free space
- **Display:** 1024x768 resolution or higher
- **Input Devices:** Keyboard and Mouse

### CODE STRUCTURE OVERVIEW:

- **Library Imports:** Uses tkinter, csv, and other built-in modules.
- **GUI Layout:** Two sections – Student Details and Marks Entry using Tkinter.
- **Buttons:** Add, Edit, Delete, Clear, Select (view-only).
- **Calculations:** Auto-calculates average, percentage, and rank.
- **Data Handling:** Saves and reads student records using CSV files.
- **Table Display:** Shows all records in a scrollable table (Treeview).
- **Event Logic:** Handles user actions like row selection and button clicks.

# CODE IMPLEMENTATION:

```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import csv

class StudentManagementApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Student Management List System")
        self.root.geometry("1150x650")

        # Frame for student details and marks side by side
        self.main_frame = tk.Frame(root)
        self.main_frame.pack(pady=10, padx=10, fill=tk.X)

        # Student Detail Frame
        self.details_frame = tk.LabelFrame(self.main_frame, text="Student Details", padx=10, pady=10)
        self.details_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Marks Frame
        self.marks_frame = tk.LabelFrame(self.main_frame, text="Marks Details", padx=10, pady=10)
        self.marks_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Student Detail Fields
        self.entries = {}
        detail_labels = ["Name", "Roll No", "DOB", "Phone"]
        for i, label in enumerate(detail_labels):
            tk.Label(self.details_frame, text=label).grid(row=i, column=0, sticky=tk.W)
            entry = tk.Entry(self.details_frame, width=20)
            entry.grid(row=i, column=1, pady=2)
            self.entries[label] = entry

        # Marks Fields
        marks_labels = ["Tamil", "English", "Maths", "Science", "Social"]
        for i, label in enumerate(marks_labels):
            tk.Label(self.marks_frame, text=label).grid(row=i, column=0, sticky=tk.W)
            entry = tk.Entry(self.marks_frame, width=10)
            entry.grid(row=i, column=1, pady=2)
            self.entries[label] = entry

        # Buttons
        self.buttons_frame = tk.Frame(root, pady=10)
        self.buttons_frame.pack()

        tk.Button(self.buttons_frame, text="Add", command=self.add_record).grid(row=0, column=0,
padx=5)
        tk.Button(self.buttons_frame, text="Edit", command=self.edit_record).grid(row=0, column=1,
padx=5)
        tk.Button(self.buttons_frame, text="Delete", command=self.delete_record).grid(row=0, column=2,
padx=5)
        tk.Button(self.buttons_frame, text="Clear", command=self.clear_fields).grid(row=0, column=3,
padx=5)
        tk.Button(self.buttons_frame, text="Save", command=self.save_to_csv).grid(row=0, column=4,
padx=5)
```

```
        # Search Section (Individual Display)
        self.search_frame = tk.LabelFrame(root, text="View Specific Student")
        self.search_frame.pack(fill=tk.X, padx=10, pady=5)
        tk.Label(self.search_frame, text="Enter Name to View:").pack(side=tk.LEFT, padx=5)
        self.search_entry = tk.Entry(self.search_frame, width=30)
        self.search_entry.pack(side=tk.LEFT, padx=5)
        tk.Button(self.search_frame,                                                    text="Select",
command=self.view_selected_student).pack(side=tk.LEFT, padx=5)

        # Individual Display Section
        self.result_frame = tk.LabelFrame(root, text="Selected Student Info (View Only)")
        self.result_frame.pack(fill=tk.X, padx=10, pady=5)
        self.result_text = tk.Text(self.result_frame, height=6, wrap=tk.WORD, state=tk.DISABLED)
        self.result_text.pack(fill=tk.BOTH, padx=5, pady=5)

        # Table Frame
        self.table_frame = tk.Frame(root)
        self.table_frame.pack(fill=tk.BOTH, expand=1)

        columns = ("Name", "Roll No", "DOB", "Phone", "Tamil", "English", "Maths", "Science", "Social",
"Average", "Percentage", "Rank")
        self.tree = ttk.Treeview(self.table_frame, columns=columns, show="headings")
        for col in columns:
            self.tree.heading(col, text=col)
            self.tree.column(col, width=90)
        self.tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Scrollbar
        scrollbar = ttk.Scrollbar(self.table_frame, orient="vertical", command=self.tree.yview)
        self.tree.configure(yscrollcommand=scrollbar.set)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

        self.tree.bind("<ButtonRelease-1>", self.select_record)
        self.selected_item = None

    def calculate_results(self, marks):
        total = sum(marks)
        avg = total / len(marks)
        percent = (total / (len(marks) * 100)) * 100
        return round(avg, 2), round(percent, 2)

    def update_ranks(self):
        all_items = [(self.tree.item(child)['values'], child) for child in self.tree.get_children()]
        sorted_items = sorted(all_items, key=lambda x: x[0][9], reverse=True)  # Average is at index 9
        for rank, (values, item_id) in enumerate(sorted_items, start=1):
            new_values = list(values)
            new_values[11] = rank  # Rank is index 11
            self.tree.item(item_id, values=new_values)

    def add_record(self):
        data = {label: entry.get() for label, entry in self.entries.items()}
        if any(v == "" for v in data.values()):
            messagebox.showwarning("Input Error", "Please fill in all fields")
            return
        try:
            marks = [int(data[subj]) for subj in ["Tamil", "English", "Maths", "Science", "Social"]]
```

```
        except ValueError:
            messagebox.showwarning("Input Error", "Please enter valid numbers for marks")
            return
        avg, percent = self.calculate_results(marks)
        row_data = [data[label] for label in ["Name", "Roll No", "DOB", "Phone", "Tamil", "English",
"Maths", "Science", "Social"]] + [avg, percent, 0]
        self.tree.insert("", "end", values=row_data)
        self.update_ranks()
        self.clear_fields()

    def select_record(self, event):
        selected = self.tree.focus()
        if selected:
            self.selected_item = selected
            values = self.tree.item(selected, 'values')
            keys = ["Name", "Roll No", "DOB", "Phone", "Tamil", "English", "Maths", "Science", "Social"]
            for i, key in enumerate(keys):
                self.entries[key].delete(0, tk.END)
                self.entries[key].insert(0, values[i])

    def edit_record(self):
        if not self.selected_item:
            messagebox.showwarning("Select Error", "Please select a row to edit")
            return
        data = {label: entry.get() for label, entry in self.entries.items()}
        if any(v == "" for v in data.values()):
            messagebox.showwarning("Input Error", "Please fill in all fields")
            return
        try:
            marks = [int(data[subj]) for subj in ["Tamil", "English", "Maths", "Science", "Social"]]
        except ValueError:
            messagebox.showwarning("Input Error", "Please enter valid numbers for marks")
            return
        avg, percent = self.calculate_results(marks)
        row_data = [data[label] for label in ["Name", "Roll No", "DOB", "Phone", "Tamil", "English",
"Maths", "Science", "Social"]] + [avg, percent, 0]
        self.tree.item(self.selected_item, values=row_data)
        self.update_ranks()
        self.clear_fields()
        self.selected_item = None

    def delete_record(self):
        selected = self.tree.focus()
        if not selected:
            messagebox.showwarning("Select Error", "Please select a row to delete")
            return
        self.tree.delete(selected)
        self.update_ranks()
        self.clear_fields()

    def clear_fields(self):
        for entry in self.entries.values():
            entry.delete(0, tk.END)
        self.selected_item = None
        self.search_entry.delete(0, tk.END)
        self.result_text.config(state=tk.NORMAL)
```

```
        self.result_text.delete("1.0", tk.END)
        self.result_text.config(state=tk.DISABLED)

    def view_selected_student(self):
        search_name = self.search_entry.get().strip().lower()
        self.result_text.config(state=tk.NORMAL)
        self.result_text.delete("1.0", tk.END)
        if not search_name:
            self.result_text.insert(tk.END, "Enter a name to view.")
        else:
            found = False
            for child in self.tree.get_children():
                values = self.tree.item(child)['values']
                if values and search_name == str(values[0]).lower():
                    found = True
                    labels = ["Name", "Roll No", "DOB", "Phone", "Tamil", "English", "Maths", "Science",
"Social", "Average", "Percentage", "Rank"]
                    for label, val in zip(labels, values):
                        self.result_text.insert(tk.END, f"{label}: {val}\n")
                    break
            if not found:
                self.result_text.insert(tk.END, f"No student found with the name '{search_name}'.")
        self.result_text.config(state=tk.DISABLED)

    def save_to_csv(self):
        file_path = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV files",
"*.csv")])
        if not file_path:
            return
        with open(file_path, mode='w', newline='') as file:
            writer = csv.writer(file)
            headers = ["Name", "Roll No", "DOB", "Phone", "Tamil", "English", "Maths", "Science",
"Social", "Average", "Percentage", "Rank"]
            writer.writerow(headers)
            for child in self.tree.get_children():
                writer.writerow(self.tree.item(child)['values'])
        messagebox.showinfo("Success", "Student data saved successfully!")

if __name__ == "__main__":
    root = tk.Tk()
    app = StudentManagementApp(root)
    root.mainloop()
```

# CODE EXPLAINATION:

## 1. Import Required Libraries:

```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import csv
```

- tkinter: For GUI components.
- ttk: For themed widgets like Treeview.
- messagebox: To show alerts and messages.
- filedialog: To open a save file dialog.
- csv: To save data in CSV format.

2. **Define Main Class:**

```
class StudentManagementApp:
```

- Creates a class for the application.

3. **Initialize GUI:**

```
def __init__(self, root):
    self.root = root
    self.root.title("Student Management List System")
    self.root.geometry("1150x650")
```

- Initializes the GUI with a title and window size.

4. **Create Main Frame:**

```
self.main_frame = tk.Frame(root)
self.main_frame.pack(pady=10, padx=10, fill=tk.X)
```

- Creates a main frame to hold student and marks sections side-by-side.

5. **Create Student Details Frame:**

```
self.details_frame = tk.LabelFrame(self.main_frame, text="Student Details", padx=10, pady=10)
self.details_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
```

- A labeled frame for inputting student personal details.

6. **Create Marks Frame:**

```
self.marks_frame = tk.LabelFrame(self.main_frame, text="Marks Details", padx=10, pady=10)

self.marks_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
```

- A labeled frame for entering subject marks.

7. **Define Entry Dictionary and Student Fields:**

```
self.entries = {}
detail_labels = ["Name", "Roll No", "DOB", "Phone"]
```

- self.entries: Stores all entry widgets.
- detail_labels: List of student personal detail fields.

8. **Create Entry Widgets for Student Details:**

```
for i, label in enumerate(detail_labels):
   tk.Label(self.details_frame, text=label).grid(row=i, column=0, sticky=tk.W)
   entry = tk.Entry(self.details_frame, width=20)
   entry.grid(row=i, column=1, pady=2)
   self.entries[label] = entry
```

- Loops through each label, creates a label and entry field, and stores it in the self.entries dictionary.

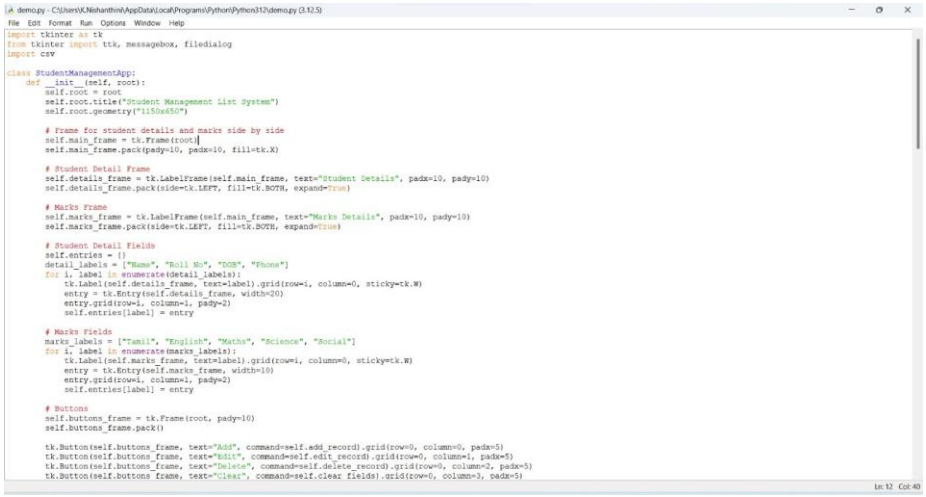
9. **Define and Create Entry for Marks:**

```
marks_labels = ["Tamil", "English", "Maths", "Science", "Social"]
for i, label in enumerate(marks_labels):
   tk.Label(self.marks_frame, text=label).grid(row=i, column=0, sticky=tk.W)
   entry = tk.Entry(self.marks_frame, width=10)
```

```
    entry.grid(row=i, column=1, pady=2)
    self.entries[label] = entry
```

- Creates entry boxes for 5 subjects and stores them.

## CODE SCREENSHOT:



```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import csv

class StudentManagementApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Student Management List System")
        self.root.geometry("1150x650")

        # Frame for student details and marks side by side
        self.main_frame = tk.Frame(root)
        self.main_frame.pack(pady=10, padx=10, fill=tk.X)

        # Student Detail Frame
        self.details_frame = tk.LabelFrame(self.main_frame, text="Student Details", padx=10, pady=10)
        self.details_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Marks Frame
        self.marks_frame = tk.LabelFrame(self.main_frame, text="Marks Details", padx=10, pady=10)
        self.marks_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Student Detail Fields
        self.entries = {}
        detail_labels = ["Name", "Roll No", "DOB", "Phone"]
        for i, label in enumerate(detail_labels):
            tk.Label(self.details_frame, text=label).grid(row=i, column=0, sticky=tk.W)
            entry = tk.Entry(self.details_frame, width=20)
            entry.grid(row=i, column=1, pady=2)
            self.entries[label] = entry

        # Marks Fields
        marks_labels = ["Tamil", "English", "Maths", "Science", "Social"]
        for i, label in enumerate(marks_labels):
            tk.Label(self.marks_frame, text=label).grid(row=i, column=0, sticky=tk.W)
            entry = tk.Entry(self.marks_frame, width=10)
            entry.grid(row=i, column=1, pady=2)
            self.entries[label] = entry

        # Buttons
        self.buttons_frame = tk.Frame(root, pady=10)
        self.buttons_frame.pack()

        tk.Button(self.buttons_frame, text="Add", command=self.add_record).grid(row=0, column=0, padx=5)
        tk.Button(self.buttons_frame, text="Edit", command=self.edit_record).grid(row=0, column=1, padx=5)
        tk.Button(self.buttons_frame, text="Delete", command=self.delete_record).grid(row=0, column=2, padx=5)
        tk.Button(self.buttons_frame, text="Clear", command=self.clear_fields).grid(row=0, column=3, padx=5)
```

## CODE OUTPUT SCREENSHOT: