CAR SHOWROOM INVENTORY SYSTEM

Project Report Submitted to

Ms. Palwinder Kaur Mangat

Submitted by

Varun Singh (25MCA20126)

In partial fulfilment for the award of the degree of

MASTER OF COMPUTER APPLICATION

IN

University Institute of Computing (UIC)

CHANDIGARH UNIVERSITY

SESSION2025-27

Abstract

The **Car Showroom Inventory System** is a Python-based application developed to streamline the management of car information, stock availability, and basic sales records. The system replaces traditional manual methods with a digital solution that supports essential **CRUD operations (Create, Read, Update, Delete)** for maintaining car data. It enables showroom staff to easily manage details such as car names, models, prices, and stock levels through a user-friendly interface.

Introduction

The Car Showroom Inventory System is a Python-based application designed to automate and simplify the management of car details, stock levels, and sales records in a showroom. Manual record-keeping often leads to errors, data misplacement, and slow processes. This system provides an efficient digital solution to store and manage information such as car name, model, price, and availability.

It allows showroom staff to easily add, update, delete, and view car data through simple operations. By maintaining accurate and organized records, the system supports better decision-making related to inventory and sales. Overall, it reduces paperwork, saves time, and improves the overall management and customer service experience within the showroom.

Modules

1. **Car Data Module:**
   Defines the structure to store car details like ID, model, company, and price.
2. **Add Car Module:**
   Allows users to add new car information into the inventory.
3. **Display Cars Module:**
   Displays all cars available in the showroom with their details.
4. **Search Car Module:**
   Searches and shows car details based on a given car ID.
5. **Exit Module:**
   Exits the system safely after completing all operations.

Why This Project Was Made

The Car Showroom Inventory System was developed to simplify and automate the process of managing car details in a showroom. Traditionally, car records were maintained manually, which often led to errors, data loss, and inefficiency. This project aims to overcome these issues by providing a computerized system, built using Python, that is fast, accurate, and easy to use.

Main Reasons:

1. **To Learn Programming Concepts:**
   To practice lists, loops, conditionals, and random number generation in Python.
2. **To Develop Logical Thinking:**
   It encourages problem-solving and algorithmic thinking while building an interactive program.
3. **To Create an Engaging Application:**
   Designed to entertain users by displaying random jokes and facts each time it runs.
4. **To Demonstrate Practical Skills:**
   Useful as a mini project to showcase understanding of coding principles and user interaction.
5. **To Build a Foundation for Future Projects:**
   The concept can be extended using file handling or APIs to fetch real-time jokes or facts.

Use of the Project

The Random Joke and Fact Generator is a simple yet interactive Python program designed for both learning and entertainment. It helps users enjoy random jokes and facts while showcasing basic programming logic.

Main Uses:

1. **Inventory Management:**
   Helps manage car details such as ID, model, company, and price efficiently.
2. **Data Accuracy:**
   Reduces manual errors by maintaining digital car records.

3. **Quick Access:**
   Allows users to easily search, add, update, or view car information.
4. **Educational Purpose:**
   Serves as a mini project for students to learn Python concepts like lists, dictionaries, functions, and control statements.
5. **Time Saving:**
   Simplifies and speeds up daily showroom operations.
6. **Future Development:**
   Can be enhanced with features like database connectivity, file storage, and sales tracking.

System Design / Methodology

Modules of the System:

1. **Car Data Module –** Defines the structure to store car details such as ID, model, company, and price.
2. **Add Car Module** – Allows the user to input and add new car information into the inventory.
3. **Display Cars Module –** Displays all the cars stored in the system with their details in a clear format.
4. **Search Car Module –** Enables the user to search for a specific car by its unique ID and view its details.
5. **Exit Module –** Safely terminates the program after completing all operations and displays a closing message.

Tools and Technologies Used

1. **Programming Language: Python** — used for implementing the main logic of the Car Showroom Inventory System.
2. **Interpreter: Python 3.x** — used to run and execute the Python program efficiently.
3. **IDE: PyCharm, Visual Studio Code, or IDLE** — used for writing, debugging, and running the code.
4. **Libraries Used:**
   a. **random** (if needed) for generating random values.
   b. **json or csv** (optional) for storing car data.
   c. **tkinter** (optional) for GUI-based implementation.
5. **Operating System:** Compatible with Windows, Linux, and macOS.
6. **Key Concepts Applied:**
   a. **Dictionaries / Classes** for storing car information.
   b. **Lists** for managing multiple car records.
   c. **Loops** for iterative operations like display and search.
   d. **Conditional Statements** for menu-driven control and validation.
   e. **Functions** for handling user choices efficiently.

Implementation / Development

Code:

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

car_inventory = []       # Main Inventory
deleted_cars = []        # Deleted Cars List
updated_cars = []        # Updated Cars List

# ---------------- Functions ----------------
def add_car():
    name = name_entry.get()
    model = model_entry.get()
    price = price_entry.get()
    stock = stock_entry.get()

    if not name or not model or not price or not stock:
        messagebox.showerror("Error", "All fields are required!")
        return

    car = {
        "name": name,
        "model": model,
        "price": float(price),
        "stock": int(stock)
    }

    car_inventory.append(car)
    update_tables()
    clear_fields()
    messagebox.showinfo("Success", "Car added successfully!")

def delete_car():
    selected = car_table.selection()
    if not selected:
        messagebox.showerror("Error", "Select a car to delete")
        return

    index = car_table.index(selected[0])
```

```
    # Add to deleted list
    deleted_cars.append(car_inventory[index].copy())

    car_inventory.pop(index)
    update_tables()
    messagebox.showinfo("Deleted", "Car deleted successfully!")

def update_stock():
    selected = car_table.selection()
    if not selected:
        messagebox.showerror("Error", "Select a car to update")
        return

    index = car_table.index(selected[0])
    new_stock = stock_entry.get()

    if not new_stock.isdigit():
        messagebox.showerror("Error", "Enter valid stock number!")
        return

    # Store old record for update history
    old = car_inventory[index].copy()
    updated_cars.append(old)

    # Update stock
    car_inventory[index]["stock"] = int(new_stock)

    update_tables()
    messagebox.showinfo("Updated", "Stock updated successfully!")

def update_tables():
    # MAIN TABLE
    for row in car_table.get_children():
        car_table.delete(row)
    for car in car_inventory:
        car_table.insert("", "end", values=(car["name"], car["model"], car["price"], car["stock"]))

    # DELETED TABLE
    for row in deleted_table.get_children():
        deleted_table.delete(row)
    for car in deleted_cars:
```

```
        deleted_table.insert("", "end", values=(car["name"], car["model"], car["price"], car["stock"]))

    # UPDATED TABLE
    for row in updated_table.get_children():
        updated_table.delete(row)
    for car in updated_cars:
        updated_table.insert("", "end", values=(car["name"], car["model"], car["price"], car["stock"]))

def clear_fields():
    name_entry.delete(0, tk.END)
    model_entry.delete(0, tk.END)
    price_entry.delete(0, tk.END)
    stock_entry.delete(0, tk.END)

# ---------------- GUI ----------------
root = tk.Tk()
root.title("Car Inventory Management System")
root.geometry("900x700")
root.config(bg="#e8f1f8")

# ---- Title ----
title = tk.Label(root,
                 text="🚗 Car Inventory Management System",
                 font=("Arial", 20, "bold"),
                 fg="#0a3d62", bg="#e8f1f8")
title.pack(pady=15)

# ---- Input Frame ----
frame = tk.Frame(root, bg="#e8f1f8")
frame.pack(pady=10)

labels = ["Car Name:", "Model:", "Price:", "Stock:"]

for i, text in enumerate(labels):
    tk.Label(frame, text=text, font=("Arial", 11, "bold"),
             bg="#e8f1f8", fg="#0a3d62").grid(row=i, column=0, sticky="w", pady=4)

name_entry = tk.Entry(frame, width=25, font=("Arial", 11))
model_entry = tk.Entry(frame, width=25, font=("Arial", 11))
price_entry = tk.Entry(frame, width=25, font=("Arial", 11))
stock_entry = tk.Entry(frame, width=25, font=("Arial", 11))
```

```
name_entry.grid(row=0, column=1, padx=10, pady=4)
model_entry.grid(row=1, column=1, padx=10, pady=4)
price_entry.grid(row=2, column=1, padx=10, pady=4)
stock_entry.grid(row=3, column=1, padx=10, pady=4)

# ---- Buttons ----
btn_frame = tk.Frame(root, bg="#e8f1f8")
btn_frame.pack(pady=10)

button_style = {
    "width": 14,
    "font": ("Arial", 11, "bold"),
    "bd": 0,
    "relief": "flat",
    "fg": "white",
    "height": 1,
    "cursor": "hand2"
}

tk.Button(btn_frame, text="Add Car", bg="#27ae60", command=add_car, **button_style).grid(row=0,
column=0, padx=8)
tk.Button(btn_frame, text="Delete Car", bg="#e74c3c", command=delete_car,
**button_style).grid(row=0, column=1, padx=8)
tk.Button(btn_frame, text="Update Stock", bg="#2980b9", command=update_stock,
**button_style).grid(row=0, column=2, padx=8)
tk.Button(btn_frame, text="Clear Fields", bg="#8e44ad", command=clear_fields,
**button_style).grid(row=0, column=3, padx=8)

# ---------------- Main Inventory Table ----------------
tk.Label(root, text="Current Inventory", bg="#e8f1f8",
         font=("Arial", 14, "bold"), fg="#0a3d62").pack()

columns = ("Name", "Model", "Price", "Stock")
car_table = ttk.Treeview(root, columns=columns, show="headings", height=8)
for col in columns:
    car_table.heading(col, text=col)
    car_table.column(col, width=150)
car_table.pack(pady=10)

# ---------------- Deleted Cars Table ----------------
```

```
tk.Label(root, text="Deleted Cars", bg="#e8f1f8",
         font=("Arial", 14, "bold"), fg="#c0392b").pack()

deleted_table = ttk.Treeview(root, columns=columns, show="headings", height=5)
for col in columns:
    deleted_table.heading(col, text=col)
    deleted_table.column(col, width=150)
deleted_table.pack(pady=10)

# ---------------- Updated Cars Table ----------------
tk.Label(root, text="Updated Cars (Stock Changed)", bg="#e8f1f8",
         font=("Arial", 14, "bold"), fg="#8e44ad").pack()

updated_table = ttk.Treeview(root, columns=columns, show="headings", height=5)
for col in columns:
    updated_table.heading(col, text=col)
    updated_table.column(col, width=150)
updated_table.pack(pady=10)

root.mainloop()
```

Coding Outputs:

Results and Discussion

The Car Showroom Inventory System successfully manages car details such as ID, model, company, and price in an organized manner. The Python program allows users to easily add new cars, display all available cars, and search for specific cars by ID. Each function works efficiently, providing accurate results and smooth user interaction.

The use of lists, dictionaries, loops, and conditional statements ensures proper data handling and reliable performance.The system operates as expected, helping to maintain an accurate car inventory while reducing manual effort. Overall, it provides a simple, effective, and efficient solution for basic car showroom management using Python.

Conclusion

The Car Showroom Inventory System project successfully demonstrates the use of fundamental Python concepts such as lists, dictionaries, loops, and conditional statements. It provides an efficient and user-friendly way to manage car records, allowing easy addition, display, and searching of car details. The system reduces manual work, minimizes errors, and improves data handling efficiency. Overall, it is a simple yet effective project that shows how basic programming logic in Python can be applied to solve real-world management problems.

References

1. **Mark Lutz**, "Learning Python", O'Reilly Media.
2. **Guido van Rossum**, "The Python Programming Language Documentation", Python.org.
3. **TutorialsPoint** — https://www.tutorialspoint.com/python/
4. **GeeksforGeeks** — https://www.geeksforgeeks.org/python-programming-language/
5. **W3Schools** — https://www.w3schools.com/python/
6. **Stack Overflow** — https://stackoverflow.com/
7. **Programiz** — https://www.programiz.com/python-programming