1.Simple ATM System using Conditions and Loops

```
def simple_atm():
    balance = 1000  # Starting balance

    while True:
        print("\nATM Menu:")
        print("1. Check Balance")
        print("2. Deposit Money")
        print("3. Withdraw Money")
        print("4. Exit")

        # Get user input for menu choice
        choice = int(input("Enter your choice (1/2/3/4): "))

        # Check the choice and perform actions
        if choice == 1:
            print(f"Your balance is: ${balance}")

        elif choice == 2:
            deposit = float(input("Enter deposit amount: $"))
            balance += deposit
            print(f"You deposited: ${deposit}. Your new balance is: ${balance}")

        elif choice == 3:
            withdraw = float(input("Enter withdrawal amount: $"))
            if withdraw <= balance:
                balance -= withdraw
                print(f"You withdrew: ${withdraw}. Your new balance is: ${balance}")
            else:
                print("Insufficient balance!")

        elif choice == 4:
            print("Thank you for using the ATM. Goodbye!")
            break  # Exit the loop

        else:
            print("Invalid choice, please try again.")

# Run the simple ATM system
simple_atm()
```

==========

2.Hotel Food Order System using Functions

```
# Function to display the menu
def display_menu():
    print("\n--- Hotel Menu ---")
    print("1. Pizza - Rs.12")
    print("2. Burger - Rs.8")
    print("3. Pasta - Rs.10")
    print("4. Salad - Rs.6")
    print("5. Exit")

# Function to take the user's order and calculate the total bill
def take_order():
    menu = {
        1: {"name": "Pizza", "price": 12},
        2: {"name": "Burger", "price": 8},
        3: {"name": "Pasta", "price": 10},
        4: {"name": "Salad", "price": 6}
    }

    total_bill = 0  # Initialize total bill

    while True:
        display_menu()  # Show the menu
        try:
            choice = int(input("Enter the item that you want to order (1-5): "))

            if choice == 5:
                print(f"Your total bill is: Rs.{total_bill}")
                print("Thank you for ordering! Goodbye!")
                break  # Exit the loop and end the program

            elif choice in menu:
                quantity = int(input(f"How many {menu[choice]['name']}s would you like to order? "))
                item_total = quantity * menu[choice]["price"]
                total_bill += item_total
                print(f"Added {quantity} {menu[choice]['name']}(s) to your order. Total for this item:
Rs.{item_total}")
            else:
                print("Invalid choice! Please select a valid menu item.")

        except ValueError:
            print("Please enter a valid number.")
```

```
# Main function to run the Hotel Food Order program
def hotel_food_order():
    print("Welcome to the Hotel Food Ordering System!")
    take_order()

# Run the program
hotel_food_order()
```

==========

3.Book Management System Using List and their Built-in Functions

```
# Function to display the list of books
def display_books(books):
    if books:
        print("\nList of Books:")
        for book in books:
            print(book)
    else:
        print("No books available in the list.")

# Function to add a book to the list
def add_book(books):
    book_name = input("Enter the name of the book to add: ")
    books.append(book_name)
    print(f"Book '{book_name}' added to the list.")

# Function to remove a book from the list
def remove_book(books):
    book_name = input("Enter the name of the book to remove: ")
    if book_name in books:
        books.remove(book_name)
        print(f"Book '{book_name}' removed from the list.")
    else:
        print(f"Book '{book_name}' not found in the list.")

# Function to sort the list of books
def sort_books(books):
    books.sort()
    print("\nBooks sorted alphabetically.")

# Function to reverse the list of books
```

```
def reverse_books(books):
    books.reverse()
    print("\nBooks list has been reversed.")

# Function to get the total number of books
def get_total_books(books):
    return len(books)

# Main function to run the Book Management System
def book_management_system():
    books = []  # Initialize an empty list of books

    while True:
        print("\n--- Book Management System ---")
        print("1. Display List of Books")
        print("2. Add a Book")
        print("3. Remove a Book")
        print("4. Sort Books Alphabetically")
        print("5. Reverse the List of Books")
        print("6. Show Total Number of Books")
        print("7. Exit")

        choice = input("Enter your choice (1-7): ")

        if choice == '1':
            display_books(books)
        elif choice == '2':
            add_book(books)
        elif choice == '3':
            remove_book(books)
        elif choice == '4':
            sort_books(books)
            display_books(books)
        elif choice == '5':
            reverse_books(books)
            display_books(books)
        elif choice == '6':
            print(f"Total number of books: {get_total_books(books)}")
        elif choice == '7':
            print("Exiting the Book Management System. Goodbye!")
            break
        else:
            print("Invalid choice! Please select a valid option.")
```

```
# Run the Book Management System
book_management_system()
```

==========

4.Flight Booking System using Tuples

```
# Tuple storing multiple flight details (Flight Number, Destination, Price)
flights = (
    ("AI202", "New York", 500),
    ("BA305", "London", 650),
    ("EK501", "Dubai", 400),
    ("SQ318", "Singapore", 550)
)

# Display available flights
print("Available Flights:")
for flight in flights:
    print(f"Flight Number: {flight[0]}, Destination: {flight[1]}, Price: ${flight[2]}")

# Simulating a ticket booking (choosing the first flight)
selected_flight = flights[0]

# Booking confirmation
print("\nFlight Ticket Booked!")
print("Flight Number:", selected_flight[0])
print("Destination:", selected_flight[1])
print("Price: $", selected_flight[2])
```

==========

5.Student Management System using Dictionary

```
# Initialize an empty dictionary to store student details
students = {}

# Menu for student management
def display_menu():
    print("\nStudent Management System")
    print("1. Add Student")
    print("2. View All Students")
    print("3. Search Student by Roll Number")
    print("4. Delete Student by Roll Number")
```

```
    print("5. Exit")

while True:
    display_menu()
    choice = input("\nEnter your choice (1-5): ")

    if choice == "1":
        # Add a student
        roll_no = input("Enter Roll Number: ")
        if roll_no in students:
            print("Roll Number already exists. Please try again.")
        else:
            name = input("Enter Student Name: ")
            marks = float(input("Enter Marks: "))
            students[roll_no] = {"name": name, "marks": marks}
            print(f"Student {name} added successfully!")

    elif choice == "2":
        # View all students
        if students:
            print("\nAll Students:")
            print("-" * 30)
            for roll_no, details in students.items():
                print(f"Roll No: {roll_no}, Name: {details['name']}, Marks: {details['marks']}")
        else:
            print("No students found.")

    elif choice == "3":
        # Search for a student
        roll_no = input("Enter Roll Number to search: ")
        if roll_no in students:
            details = students[roll_no]
            print(f"Roll No: {roll_no}, Name: {details['name']}, Marks: {details['marks']}")
        else:
            print("Student not found.")

    elif choice == "4":
        # Delete a student
        roll_no = input("Enter Roll Number to delete: ")
        if roll_no in students:
            del students[roll_no]
            print("Student deleted successfully.")
        else:
            print("Student not found.")
```

```
    elif choice == "5":
        # Exit the program
        print("Exiting the Student Management System. Goodbye!")
        break

    else:
        print("Invalid choice. Please try again.")


=======================================
```

6. Age Processing using apply(), filter(), map(), and reduce() functions.

```
import pandas as pd
from functools import reduce

# Sample data (age of people in a list)
ages = [15, 22, 19, 35, 40, 60, 55, 30, 65, 72, 80]

# 1. Using `apply()` with pandas to categorize people into age groups (Child, Adult, Senior)
df = pd.DataFrame({'Age': ages})

def categorize_age(age):
    if age < 18:
        return 'Child'
    elif 18 <= age <= 65:
        return 'Adult'
    else:
        return 'Senior'

df['Age Group'] = df['Age'].apply(categorize_age)
print("DataFrame after applying categorize_age function:")
print(df)

# 2. Using `filter()` to filter out ages that are less than 18 (child ages)
children_ages = list(filter(lambda age: age < 18, ages))
print("\nAges of children:")
print(children_ages)

# 3. Using `map()` to increase everyone's age by 1 year (for the next birthday)
next_birthday_ages = list(map(lambda age: age + 1, ages))
print("\nAges after increasing by 1 (next birthday):")
print(next_birthday_ages)
```

```
# 4. Using `reduce()` to find the total sum of all ages
total_age = reduce(lambda x, y: x + y, ages)
print("\nTotal sum of all ages:")
print(total_age)

# 5. Optional: Using reduce to find the average age by dividing the total sum by the number of
people
average_age = total_age / len(ages)
print("\nAverage age of all people:")
print(average_age)

=====================================
```

7. Implementing a Simple Contact Book by using Modules

```
import os
import sys

CONTACTS_FILE = "contacts.txt"  # File to store contacts

# Load contacts from file
def load_contacts():
    if os.path.exists(CONTACTS_FILE):
        with open(CONTACTS_FILE, "r") as file:
            return [line.strip() for line in file.readlines()]
    return []

# Save contacts to file
def save_contacts(contacts):
    with open(CONTACTS_FILE, "w") as file:
        file.writelines("\n".join(contacts))

# Add a new contact
def add_contact():
    name = input("Enter contact name: ")
    phone = input("Enter contact number: ")

    contact = f"{name}: {phone}"
    contacts = load_contacts()
    contacts.append(contact)
    save_contacts(contacts)

    print(f"Contact '{name}' added successfully!")
```

```
# View all contacts
def view_contacts():
    contacts = load_contacts()
    if not contacts:
        print("No contacts available.")
    else:
        print("\nContact List:")
        for idx, contact in enumerate(contacts, start=1):
            print(f"{idx}. {contact}")

# Delete a contact
def delete_contact():
    view_contacts()
    contacts = load_contacts()

    if not contacts:
        return

    try:
        contact_number = int(input("Enter contact number to delete: ")) - 1
        if 0 <= contact_number < len(contacts):
            removed_contact = contacts.pop(contact_number)
            save_contacts(contacts)
            print(f"Deleted contact: {removed_contact}")
        else:
            print("Invalid contact number.")
    except ValueError:
        print("Please enter a valid number.")

# Display menu
def display_menu():
    print("\nContact Book")
    print("1. Add Contact")
    print("2. View Contacts")
    print("3. Delete Contact")
    print("4. Exit")

# Main program loop
while True:
    display_menu()
    choice = input("\nEnter your choice (1-4): ")

    if choice == "1":
```

```
        add_contact()
    elif choice == "2":
        view_contacts()
    elif choice == "3":
        delete_contact()
    elif choice == "4":
        print("Exiting Contact Book. Goodbye!")
        sys.exit()
    else:
        print("Invalid choice. Please try again.")
```

==========

8. Car details Processing using Classes and Instances

```
# Define the Car class
class Car:
    def __init__(self, make, model, year):
        """Initialize the car with make, model, and year."""
        self.make = make
        self.model = model
        self.year = year
        self.engine_started = False  # Engine is initially off

    def start_engine(self):
        """Start the engine of the car."""
        if not self.engine_started:
            self.engine_started = True
            print(f"The engine of the {self.year} {self.make} {self.model} has started.")
        else:
            print("The engine is already running.")

    def stop_engine(self):
        """Stop the engine of the car."""
        if self.engine_started:
            self.engine_started = False
            print(f"The engine of the {self.year} {self.make} {self.model} has stopped.")
        else:
            print("The engine is already off.")

    def car_info(self):
        """Display information about the car."""
        status = "running" if self.engine_started else "off"
```

```
        print(f"{self.year} {self.make} {self.model} (Engine: {status})")

# Create instances (objects) of the Car class
car1 = Car("Toyota", "Corolla", 2020)
car2 = Car("Honda", "Civic", 2022)

# Perform some actions on car1
car1.start_engine()  # Start the engine of car1
car1.car_info()  # Display car1's info
car1.stop_engine()  # Stop the engine of car1
car1.car_info()  # Display car1's info

# Perform some actions on car2
car2.start_engine()  # Start the engine of car2
car2.car_info()  # Display car2's info
```

==========

9. Password Validation using Regular Expression

```
import re

# Function to validate password
def validate_password(password):
    # Regular expression pattern to check the password conditions
    # ^: start of the string
    # (?=.*[A-Z]): at least one uppercase letter
    # (?=.*[a-z]): at least one lowercase letter
    # (?=.*\d): at least one digit
    # (?=.*[!@#$%^&*()_+]): at least one special character
    # .{8,}: the length should be at least 8 characters
    pattern = r'^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[!@#$%^&*()_+]).{8,}$'

    if re.match(pattern, password):
        return True
    else:
        return False

# Sample passwords to test
passwords = [
    "Password123!",   # valid
    "password",       # invalid: no uppercase, no special character
    "PASSWORD123!",   # invalid: no lowercase letter
    "Pass123",        # invalid: too short
```

```
    "P@ssw0rd!",      # valid
    "12345678",       # invalid: no uppercase, no special character
]

# Check and validate each password
for password in passwords:
    if validate_password(password):
        print(f"'{password}' is a valid password.")
    else:
        print(f"'{password}' is an invalid password.")
```

==========

10. Basic Expense Manager with File Storage

```
import os

EXPENSE_FILE = "expenses.txt"  # File to store expenses

# Function to add an expense
def add_expense():
    date = input("Enter date (YYYY-MM-DD): ")
    category = input("Enter category (Food, Transport, etc.): ")
    amount = input("Enter amount: ")

    with open(EXPENSE_FILE, "a") as file:  # Append mode
        file.write(f"{date}, {category}, {amount}\n")
    print("Expense added successfully!")

# Function to view expenses
def view_expenses():
    if os.path.exists(EXPENSE_FILE):  # Check if file exists
        with open(EXPENSE_FILE, "r") as file:
            expenses = file.readlines()
        if expenses:
            print("\nYour Expenses:")
            for idx, expense in enumerate(expenses, start=1):
                print(f"{idx}. {expense.strip()}")
        else:
            print("No expenses found.")
    else:
        print("No expenses found.")

# Function to delete all expenses
```

```
def delete_expenses():
    if os.path.exists(EXPENSE_FILE):
        os.remove(EXPENSE_FILE)  # Delete file
        print("All expenses deleted successfully!")
    else:
        print("No expenses to delete.")

# Main menu loop
while True:
    print("\nExpense Tracker Application")
    print("1. Add Expense")
    print("2. View Expenses")
    print("3. Delete All Expenses")
    print("4. Exit")

    choice = input("Enter choice: ")

    if choice == "1":
        add_expense()
    elif choice == "2":
        view_expenses()
    elif choice == "3":
        delete_expenses()
    elif choice == "4":
        print("Exiting. Goodbye!")
        break
    else:
        print("Invalid choice. Try again.")
```

==========

11. Simple Employee Details using strings and their built-in functions

```
# Function to perform string operations on employee details
def employee_string_operations():
    # Take basic employee details as input
    employee_name = input("Enter the employee's name: ").strip()
    employee_id = input("Enter the employee's ID: ").strip()
    department = input("Enter the employee's department: ").strip()

    print("\n--- Employee Details ---")
    print(f"Employee Name: {employee_name}")
    print(f"Employee ID: {employee_id}")
    print(f"Department: {department}")
```

```
    # 1. Count occurrences of a substring in the name (e.g., 'a')
    substring = input("Enter a substring to count in the name: ").strip()
    count = employee_name.lower().count(substring.lower())  # case insensitive
    print(f"The substring '{substring}' appears {count} times in the name.")

    # 2. Check if the name is alphanumeric
    is_alphanumeric = employee_name.isalnum()
    print(f"Is the name alphanumeric? {is_alphanumeric}")

    # 3. Convert the name to uppercase
    print(f"Name in uppercase: {employee_name.upper()}")

    # 4. Convert the name to lowercase
    print(f"Name in lowercase: {employee_name.lower()}")

    # 5. Replace a part of the name (e.g., 'John' with 'Jon')
    old_substring = input("Enter part of the name to replace: ").strip()
    new_substring = input("Enter the new name part: ").strip()
    modified_name = employee_name.replace(old_substring, new_substring)
    print(f"Name after replacement: {modified_name}")

    # 6. Find the position of a substring in the name
    position = employee_name.lower().find(substring.lower())
    if position != -1:
        print(f"The substring '{substring}' is found at position {position}.")
    else:
        print(f"The substring '{substring}' is not found in the name.")

    # 7. Trim whitespace from the name
    trimmed_name = employee_name.strip()
    print(f"Name after trimming whitespace: {trimmed_name}")


# Run the program
if __name__ == "__main__":
    employee_string_operations()


=====================================
```

12. Dynamic Student Data Insertion Using SQLite Database

```
import sqlite3
```

```
import pandas as pd

# Connect to SQLite database (Creates if not exists)
conn = sqlite3.connect("students.db")
cursor = conn.cursor()

# Create table if it doesn't exist
cursor.execute('''
    CREATE TABLE IF NOT EXISTS students (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        age INTEGER NOT NULL
    )
''')
conn.commit()

print("Database and table created successfully!")

# Function to insert dynamic data
def insert_student(name, age):
    cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", (name, age))
    conn.commit()
    print(f"Student '{name}' added successfully!")

# Insert multiple students dynamically
n = int(input("How many students do you want to add? "))

for _ in range(n):
    name = input("Enter student name: ")
    age = int(input("Enter student age: "))
    insert_student(name, age)

# View all students
df = pd.read_sql("SELECT * FROM students", conn)
print("\nStudent Records:")
print(df)

# Close the database connection
conn.close()
print("Database connection closed.")
```

==========

13. Predicting Customer Shopping Behavior Using Pattern Recognition

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Load dataset from CSV
df = pd.read_csv("customer_data.csv")  # Ensure the CSV file is in the same directory

# Display first few rows of data
print(df.head())

# Encode categorical target variable (High = 2, Medium = 1, Low = 0)
label_encoder = LabelEncoder()
df["Spending_Category"] = label_encoder.fit_transform(df["Spending_Category"])

# Separate features (X) and target (y)
X = df.drop(columns=["Spending_Category"])  # Features (Age, Income, Shopping Score)
y = df["Spending_Category"]  # Target (0 = Low, 1 = Medium, 2 = High)

# Split dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features for better accuracy
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train a k-NN model (k=5)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predict on test data
y_pred = knn.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")

# Confusion matrix visualization
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, cmap="Blues", xticklabels=["Low","Medium","High"],
yticklabels=["Low", "Medium", "High"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

=====================================
```