Qn: Develop a program for extracting product names, prices, and ratings from a product

listing page. Verify the CSV file to ensure the data is correctly formatted.

```
pip install requests beautifulsoup4 pandas

import requests

from bs4 import BeautifulSoup

import pandas as pd


# URL of the product listing page

url = 'https://example.com/products'


# Fetch the webpage content

response = requests.get(url)

if response.status_code != 200:

   raise Exception(f"Failed to load page {url}")


# Parse the content with BeautifulSoup

soup = BeautifulSoup(response.text, 'html.parser')


# Define lists to store extracted data

product_names = []

prices = []

ratings = []


# Replace the selectors below with those specific to your page structure

for product in soup.select('.product-item'):

   # Extract product name

   name = product.select_one('.product-name')
```

```
    product_names.append(name.text.strip() if name else 'N/A')

    # Extract product price
    price = product.select_one('.product-price')
    prices.append(price.text.strip() if price else 'N/A')

    # Extract product rating
    rating = product.select_one('.product-rating')
    ratings.append(rating.text.strip() if rating else 'N/A')

# Create a DataFrame
df = pd.DataFrame({
    'Product Name': product_names,
    'Price': prices,
    'Rating': ratings
})

# Save to CSV
csv_file = 'products.csv'
df.to_csv(csv_file, index=False)

# Verify CSV file format
def verify_csv(file_path):
    try:
        df = pd.read_csv(file_path)
        assert all(column in df.columns for column in ['Product Name', 'Price', 'Rating']),
"Column mismatch"
        print("CSV file verified and correctly formatted.")
    except Exception as e:
        print(f"Error in CSV format: {e}")

# Run verification
verify_csv(csv_file)
```

Qn: Build a python program to create a sequence in an RPA tool that collects user inputs

(like name, age, and principal amount, interest rate, time period) and then displays simple

interest using a message box.

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

# Function to calculate simple interest
def calculate_simple_interest(principal, rate, time):
    return (principal * rate * time) / 100

# Function to get user inputs and display the simple interest
def get_user_inputs():
    # Initialize tkinter root window
    root = tk.Tk()
    root.withdraw()  # Hide the root window

    # Get user inputs using simple dialogs
    name = simpledialog.askstring("Input", "Enter your name:")
    age = simpledialog.askinteger("Input", "Enter your age:")
    principal = simpledialog.askfloat("Input", "Enter the principal amount:")
    rate = simpledialog.askfloat("Input", "Enter the interest rate (in %):")
    time = simpledialog.askfloat("Input", "Enter the time period (in years):")
    # Calculate the simple interest
    simple_interest = calculate_simple_interest(principal, rate, time)

    # Display the result in a message box
    messagebox.showinfo("Simple Interest Calculation",
                f"Hello, {name}!\n"
                f"Your age: {age}\n"
                f"Principal Amount: {principal}\n"
                f"Interest Rate: {rate}%\n"
                f"Time Period: {time} years\n"
                f"Simple Interest: {simple_interest:.2f}")

# Run the function
get_user_inputs()
```

Qn: Build a python program to create a sequence in an RPA tool that collects user inputs

(like user rating (1-5 stars),comments, would they recommend the product (Yes/No)) and

then displays these using a message box.

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

# Function to get user inputs and display them
def get_user_feedback():
    # Initialize tkinter root window
    root = tk.Tk()
    root.withdraw()  # Hide the root window
    # Get user inputs using simple dialogs
    rating = simpledialog.askinteger("Input", "Please rate the product (1-5 stars):",
minvalue=1, maxvalue=5)
    comments = simpledialog.askstring("Input", "Enter your comments about the
product:")
    recommend = simpledialog.askstring("Input", "Would you recommend this
product? (Yes/No):")

    # Display the collected feedback in a message box
    messagebox.showinfo("User Feedback",
                        f"Rating: {rating} stars\n"
                        f"Comments: {comments}\n"
                        f"Recommend: {recommend}")

# Run the function
get_user_feedback()
```

Qn: Build a program for validating data in an Excel spreadsheet using UI automation. Use

Excel Application Scope to open an Excel file.

```
pip install openpyxl
import openpyxl
from openpyxl.styles import PatternFill

# Open the Excel file
def validate_excel_data(file_path):
    workbook = openpyxl.load_workbook(file_path)
    sheet = workbook.active  # Select the active sheet

    # Define validation rules for columns
```

```
  validations = {
    'A': {'type': 'non_empty', 'description': 'Name should not be empty'},
    'B': {'type': 'numeric', 'description': 'Age should be a number'},
    'C': {'type': 'non_empty', 'description': 'Email should not be empty'}
  }

  # Style for invalid cells (red fill)
  red_fill = PatternFill(start_color="FF0000", end_color="FF0000", fill_type="solid")

  errors = []

  # Iterate through each row and validate cells based on rules
  for row in sheet.iter_rows(min_row=2, max_row=sheet.max_row):  # Skip header
row
    for cell in row:
      column_letter = cell.column_letter
      if column_letter in validations:
        rule = validations[column_letter]

        # Apply validation rules
        if rule['
```

Qn: Develop a flowchart that checks for a specific condition (e.g., if a file exists).

```
import os

# Input the file path you want to check
file_path = input("Enter the file path: ")

# Check if the file exists
```

```
if os.path.exists(file_path):
   print("File found. You can open or process the file.")
   # Place code here to open or process the file if needed
else:
   print("File not found. Please check the file path and try again.")
```

Qn: Develop a program for gathering data from different applications and compile a report. Use UI automation activities like Click, Type Into, and Get Text to gather data from each application.

```
import pyautogui
import pandas as pd
import time

# Function to gather data from Application 1
def gather_data_from_app1():
   pyautogui.click(x=100, y=200)  # Click to focus on Application 1
   time.sleep(1)
   pyautogui.typewrite("Search Term\n")  # Type into search box and press Enter
   time.sleep(2)
   data1 = pyautogui.getActiveWindowText()  # Get text from a selected area
(hypothetical method)
   return data1

# Function to gather data from Application 2
def gather_data_from_app2():
   pyautogui.click(x=200, y=300)  # Click to focus on Application 2
   time.sleep(1)
   pyautogui.typewrite("Another Search Term\n")
   time.sleep(2)
```

```
    data2 = pyautogui.getActiveWindowText()  # Get text from a selected area
(hypothetical method)
    return data2


# Collect data from both applications
data1 = gather_data_from_app1()
data2 = gather_data_from_app2()


# Compile data into a DataFrame for the report
df = pd.DataFrame({
    'Application': ['App1', 'App2'],
    'Data': [data1, data2]
})


# Save the report as an Excel file
df.to_excel("DataReport.xlsx", index=False)
print("Report saved as DataReport.xlsx")
```

Qn: Build a workflow for managing support tickets based on user feedback. Set up states

for "Ticket Created," "In Progress," "Resolved," and "Closed."

```
class SupportTicket:
    def __init__(self, ticket_id, description):
        self.ticket_id = ticket_id
        self.description = description
        self.state = "Ticket Created"


    def progress(self):
        if self.state == "Ticket Created":
            self.state = "In Progress"
        elif self.state == "In Progress":
            self.state = "Resolved"
        elif self.state == "Resolved":
            self.state = "Closed"
        else:
            print("Ticket is already closed.")

    def reopen(self):
        if self.state == "Resolved":
            self.state = "In Progress"
        else:
            print("Only resolved tickets can be reopened.")

    def __str__(self):
        return f"Ticket {self.ticket_id} - {self.state}: {self.description}"


# Example usage
ticket = SupportTicket(1, "User cannot access the dashboard")

print(ticket)          # Ticket Created
ticket.progress()      # Move to "In Progress"
print(ticket)

ticket.progress()      # Move to "Resolved"
print(ticket)

ticket.reopen()        # Reopen and move back to "In Progress"
```

```
print(ticket)


ticket.progress()     # Move to "Resolved" again
ticket.progress()     # Finally move to "Closed"
print(ticket)
```

Qn: Develop a program for scraping the latest scores and match details from a sports

website. Check the CSV file to confirm that the scores and details are accurate and

complete.

```
pip install requests beautifulsoup4 pandas
import requests
from bs4 import BeautifulSoup
import pandas as pd


# Function to scrape sports scores from a website
def scrape_scores():
  url = "https://www.example-sports-website.com"  # Replace with the actual
sports website URL
  response = requests.get(url)
  soup = BeautifulSoup(response.text, 'html.parser')


  match_details = []


  # Example: Assuming match details are stored in a specific HTML structure
  # Replace with actual HTML elements and class names of the sports website
  matches = soup.find_all('div', class_='match-details')  # Adjust according to the
website's structure


  for match in matches:
        team1 = match.find('span', class_='team1').text.strip()
        team2 = match.find('span', class_='team2').text.strip()
        score1 = match.find('span', class_='score1').text.strip()
        score2 = match.find('span', class_='score2').text.strip()
        match_time = match.find('span', class_='match-time').text.strip()

        match_details.append([team1, team2, score1, score2, match_time])

    return match_details

# Function to save scraped data to a CSV file
def save_to_csv(match_details, filename="sports_scores.csv"):
    df = pd.DataFrame(match_details, columns=['Team 1', 'Team 2', 'Score 1', 'Score
2', 'Match Time'])
    df.to_csv(filename, index=False)

# Function to check if the CSV file is accurate and complete
def check_csv(filename="sports_scores.csv"):
    try:
        df = pd.read_csv(filename)

        # Check if there are any missing or invalid entries
        if df.isnull().values.any():
            print("CSV contains missing data.")
        else:
            print("CSV is complete and has no missing data.")

        # Check if scores are valid (non-empty and numerical)
        for index, row in df.iterrows():
            if not row['Score 1'].isdigit() or not row['Score 2'].isdigit():
                print(f"Invalid score data at row {index + 1}")

    except FileNotFoundError:
        print(f"{filename} not found.")
    except Exception as e:
        print(f"An error occurred while checking the CSV: {e}")

# Main function to scrape, save, and check
def main():
    match_details = scrape_scores()
    save_to_csv(match_details)
    check_csv()

# Run the program
main()
```

Qn: Develop a program to open an application from the system and automate to write a

line of text in it.

```
import pyautogui
import subprocess
import time

# Function to open an application (Notepad as an example)
def open_application():
    # Open Notepad (can replace with any other application path)
    subprocess.Popen(['notepad.exe'])
    time.sleep(2)  # Wait for the application to open


# Function to write a line of text in the opened application

def write_text_in_application():

    pyautogui.write("Hello, this is an automated line of text.", interval=0.1)


# Main function to open application and write text

def main():

    open_application()

    write_text_in_application()


# Run the program

main()
```

Qn: Develop a program for reading data from one Excel sheet, manipulate it, and write it

to another sheet. Utilize arguments to pass the input and output DataTable between

workflows

```
import pandas as pd


# Function to read data from an Excel sheet

def read_data(input_file):

    df = pd.read_excel(input_file)

    return df


# Function to manipulate the data (example: adding a new column)

def manipulate_data(df):

    df['New Column'] = df['Existing Column'] * 2  # Example manipulation

    return df
```

```
# Function to write data to another Excel sheet

def write_data(output_file, df):

   df.to_excel(output_file, index=False)


# Main function to manage the workflow

def main(input_file, output_file):

   df = read_data(input_file)

   df = manipulate_data(df)

   write_data(output_file, df)


# Example usage

input_file = 'input_data.xlsx'

output_file = 'output_data.xlsx'

main(input_file, output_file)
```

Qn: evelop a program for collecting user input, validate it, and provide feedback. Use

arguments to allow the input prompt message to be customized.

```
def get_user_input(prompt_message):

   # Collect user input

   user_input = input(prompt_message)


   # Validate the input (Example: Check if input is a number)

   if user_input.isdigit():

      return int(user_input)

   else:

      print("Invalid input. Please enter a valid number.")

      return None
```

```
def main(prompt_message):
    # Collect and validate user input
    valid_input = None
    while valid_input is None:
        valid_input = get_user_input(prompt_message)

    # Provide feedback
    print(f"Thank you! You entered a valid number: {valid_input}")

# Example usage
prompt_message = "Please enter a number: "
main(prompt_message)
```

Qn: Develop a flowchart that prompts the user to select a report type (e.g., Sales, Inventory, Customer

```
def display_report_options():
    print("Select a report type:")
    print("1. Sales Report")
    print("2. Inventory Report")
    print("3. Customer Report")

def generate_sales_report():
    print("Generating Sales Report...")

def generate_inventory_report():
    print("Generating Inventory Report...")

def generate_customer_report():
    print("Generating Customer Report...")


def main():
    display_report_options()
    choice = input("Enter the number corresponding to your selection: ")


    if choice == "1":
      generate_sales_report()
    elif choice == "2":
      generate_inventory_report()
    elif choice == "3":
      generate_customer_report()
    else:
      print("Invalid selection. Please choose a valid option.")


if __name__ == "__main__":
    main()
```

Qn: Develop a game where the user tries to guess a randomly generated number. Use a

state machine with states for "Guessing," "Too High," "Too Low," and "Correct Guess."

```
import random


class GuessingGame:
  def __init__(self):
    self.target_number = random.randint(1, 100)
    self.state = "Guessing"
    self.attempts = 0
    def make_guess(self, guess):
        self.attempts += 1

        if self.state == "Guessing":
            if guess < self.target_number:
                self.state = "Too Low"
                print("Too low!")
            elif guess > self.target_number:
                self.state = "Too High"
                print("Too high!")
            else:
                self.state = "Correct Guess"
                print(f"Correct! You guessed the number in {self.attempts} attempts.")
        else:
            print("Game over! Please restart the game.")

    def restart_game(self):
        self.target_number = random.randint(1, 100)
        self.state = "Guessing"
        self.attempts = 0
        print("Game restarted! Start guessing again.")

def main():
    game = GuessingGame()

    while game.state != "Correct Guess":
        try:
            guess = int(input("Enter your guess (between 1 and 100): "))
            game.make_guess(guess)
        except ValueError:
            print("Please enter a valid integer.")

    if game.state == "Correct Guess":
        restart = input("Do you want to play again? (yes/no): ").lower()
        if restart == 'yes':
            game.restart_game()
            main()

if __name__ == "__main__":
    main()
```

Qn: Build a python program to create a sequence in an RPA tool that collects user inputs

(like name, age, and email) and then displays these inputs using a message box.

```
import pyautogui
import tkinter as tk
from tkinter import messagebox

# Function to collect user inputs
def collect_user_inputs():
    # Create a simple Tkinter window to collect inputs
    window = tk.Tk()
    window.title("User Input Collection")

    # Name input
    name_label = tk.Label(window, text="Enter your name:")
    name_label.pack(pady=5)
    name_entry = tk.Entry(window)
    name_entry.pack(pady=5)

    # Age input
    age_label = tk.Label(window, text="Enter your age:")
    age_label.pack(pady=5)
    age_entry = tk.Entry(window)
    age_entry.pack(pady=5)

    # Email input
    email_label = tk.Label(window, text="Enter your email:")
    email_label.pack(pady=5)
    email_entry = tk.Entry(window)
    email_entry.pack(pady=5)

    # Function to process and display inputs
    def on_submit():
        name = name_entry.get()
        age = age_entry.get()
        email = email_entry.get()

        # Display the collected inputs in a message box
        messagebox.showinfo("User Inputs", f"Name: {name}\nAge: {age}\nEmail:
{email}")
        window.destroy()  # Close the window after submission

    # Submit button
    submit_button = tk.Button(window, text="Submit", command=on_submit)
    submit_button.pack(pady=20)


    window.mainloop()


# Main function to execute the RPA sequence
def main():
    collect_user_inputs()


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

Qn: Build a program for retrieving emails, extract specific information, and log it. Create

arguments to allow passing the folder name as input and logging details as output.

```
import imaplib
import email
from email.header import decode_header
import logging


# Function to set up logging
def setup_logging(output_log_file):
    logging.basicConfig(filename=output_log_file,
                        level=logging.INFO,
                        format='%(asctime)s - %(message)s')


# Function to retrieve emails from a folder
def retrieve_emails(imap_server, email_user, email_pass, folder_name):
    try:
```

```
# Connect to the email server
mail = imaplib.IMAP4_SSL(imap_server)
mail.login(email_user, email_pass)

# Select the folder (INBOX by default)
mail.select(folder_name)

# Search for all emails in the folder
status, messages = mail.search(None, 'ALL')

# Get the list of email IDs
email_ids = messages[0].split()
logging.info(f"Found {len(email_ids)} emails in folder '{folder_name}'")

for email_id in email_ids:
    # Fetch each email by ID
    status, msg_data = mail.fetch(email_id, '(RFC822)')
    for response_part in msg_data:
        if isinstance(response_part, tuple):
            msg = email.message_from_bytes(response_part[1])

            # Decode the email subject
            subject, encoding = decode_header(msg["Subject"])[0]
            if isinstance(subject, bytes):
                subject = subject.decode(encoding or 'utf-8')

            # Get the sender
            from_ = msg.get("From")
            # Get the date
            date = msg.get("Date")

            # Log the extracted email details
            logging.info(f"Subject: {subject}, From: {from_}, Date: {date}")

        mail.logout()

    except Exception as e:
        logging.error(f"An error occurred: {e}")

# Main function to execute the workflow
def main(imap_server, email_user, email_pass, folder_name, output_log_file):
    setup_logging(output_log_file)
    retrieve_emails(imap_server, email_user, email_pass, folder_name)

if __name__ == "__main__":
    # Example credentials and folder (can be replaced or passed as arguments)
    imap_server = "imap.gmail.com"
    email_user = "your_email@gmail.com"
    email_pass = "your_password"
    folder_name = "INBOX"  # Folder name (could be 'Sent', 'Spam', etc.)
    output_log_file = "email_log.txt"  # Log file name

    main(imap_server, email_user, email_pass, folder_name, output_log_file)
```

Qn: Build a program for extracting current weather data from a weather website.
Review

the CSV file to ensure all data points are captured correctly.

```
import requests

from bs4 import BeautifulSoup

import pandas as pd

import csv


# Function to scrape weather data from the website

def scrape_weather_data(url):

  response = requests.get(url)

  soup = BeautifulSoup(response.content, "html.parser")


  # Example placeholders: update these selectors to match the actual website
structure

  location = soup.find("h1", class_="current-location").get_text(strip=True)

  temperature = soup.find("span", class_="current-temp").get_text(strip=True)

  condition = soup.find("div", class_="current-weather").get_text(strip=True)

  humidity = soup.find("span", class_="humidity").get_text(strip=True)

  wind_speed = soup.find("span", class_="wind-speed").get_text(strip=True)


  # Collect the data into a dictionary

  weather_data = {

    "Location": location,

    "Temperature": temperature,

    "Condition": condition,

    "Humidity": humidity,

    "Wind Speed": wind_speed,

  }
  return weather_data


# Function to save the data to a CSV file
def save_to_csv(weather_data, filename="weather_data.csv"):
  df = pd.DataFrame([weather_data])
  df.to_csv(filename, index=False)


# Function to check the CSV file for data integrity
def check_csv(filename="weather_data.csv"):
  try:
    df = pd.read_csv(filename)


    # Check for missing values
    if df.isnull().values.any():
      print("CSV contains missing data.")
    else:
      print("CSV is complete and all data points are captured correctly.")


    # Print the captured data for review
    print("\nCaptured Weather Data:\n", df)


  except FileNotFoundError:
    print(f"{filename} not found.")
  except Exception as e:
    print(f"An error occurred while checking the CSV: {e}")


# Main function to run the weather scraping and CSV check
def main():
  # Example weather website URL (replace with actual website URL)
  url = "https://www.example-weather-website.com/current"

  # Scrape weather data
  weather_data = scrape_weather_data(url)

  # Save to CSV
  save_to_csv(weather_data)

  # Check CSV integrity
  check_csv()

if __name__ == "__main__":
  main()
```

Qn: Develop a flowchart that includes a form submission process.

```
# Import required modules
from tkinter import Tk, Label, Entry, Button, messagebox

# Function to validate form data
def validate_form_data(name, email, age):
    if not name or not email or not age:
        return False
    if not age.isdigit():
        return False
    return True

# Function to submit form
def submit_form():
    name = name_entry.get()
    email = email_entry.get()
    age = age_entry.get()

    # Validate form data
    if validate_form_data(name, email, age):
        # Form data is valid, display confirmation message
        messagebox.showinfo("Success", "Form submitted successfully!")
    else:
        # Form data is invalid, display error message
        messagebox.showerror("Error", "Please fill all fields correctly.")

# Create form UI using Tkinter
app = Tk()
app.title("Form Submission")

Label(app, text="Name:").grid(row=0, column=0, padx=10, pady=10)
name_entry = Entry(app)
name_entry.grid(row=0, column=1, padx=10, pady=10)

Label(app, text="Email:").grid(row=1, column=0, padx=10, pady=10)
email_entry = Entry(app)
email_entry.grid(row=1, column=1, padx=10, pady=10)

Label(app, text="Age:").grid(row=2, column=0, padx=10, pady=10)
age_entry = Entry(app)
age_entry.grid(row=2, column=1, padx=10, pady=10)
```

```
submit_button = Button(app, text="Submit", command=submit_form)
submit_button.grid(row=3, columnspan=2, pady=20)

# Run the app
app.mainloop()
```

Qn: Build a program showing the use of UI automation to fill out a web form automatically. (Launch a web browser and navigate to a sample form (e.g., a contact form))

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time

# Define the form data
form_data = {
  "name": "John Doe",
  "email": "john.doe@example.com",
  "subject": "Automated Form Submission",
  "message": "This is a test message filled automatically by Selenium."
}

# Initialize the web driver (make sure to have ChromeDriver installed and in PATH)
driver = webdriver.Chrome()

# Navigate to the sample contact form page
driver.get("https://www.example.com/contact-form")
```

```
# Wait for the page to load
time.sleep(2)

# Fill out the form fields (modify the selectors as per the actual form structure)
try:
    name_field = driver.find_element(By.NAME, "name")  # Replace with actual name attribute
    email_field = driver.find_element(By.NAME, "email")  # Replace with actual name attribute
    subject_field = driver.find_element(By.NAME, "subject")  # Replace with actual name attribute
    message_field = driver.find_element(By.NAME, "message")  # Replace with actual name attribute

    # Fill each field with the data from the dictionary
    name_field.send_keys(form_data["name"])
    email_field.send_keys(form_data["email"])
    subject_field.send_keys(form_data["subject"])
    message_field.send_keys(form_data["message"])

    # Submit the form (adjust if the form uses a different button type)
    submit_button = driver.find_element(By.CSS_SELECTOR, "button[type='submit']")
    submit_button.click()

    print("Form submitted successfully!")

except Exception as e:
    print("An error occurred while filling the form:", e)
```

```
finally:
    # Wait and then close the browser
    time.sleep(5)
    driver.quit()
```

Qn: Develop a flowchart that prompts the user to select their role (e.g., Admin, User,

Guest).

```
def display_role_options():
    print("Select your role:")
    print("1. Admin")
    print("2. User")
    print("3. Guest")
    print("Enter the number corresponding to your role:")


def main():
    display_role_options()
    choice = input("Role: ")


    if choice == '1':
        print("Welcome, Admin! You have full access.")
    elif choice == '2':
        print("Welcome, User! You have limited access.")
    elif choice == '3':
        print("Welcome, Guest! You have minimal access.")
    else:
        print("Invalid selection. Please choose a valid role.")


if __name__ == "__main__":
    main()
```

Qn: Develop a simple quiz that provides feedback based on user answers. Create states for

"Question 1," "Question 2," "Correct Answer," and "Wrong Answer."

```
def question_1():
    print("Question 1: What is the capital of France?")
    answer = input("Your answer: ").strip().lower()
    if answer == "paris":
        correct_answer()
        question_2()  # Move to the next question
    else:
        wrong_answer()
        question_1()  # Repeat the question if the answer is incorrect


def question_2():
    print("Question 2: What is 5 + 7?")
    answer = input("Your answer: ").strip()
    if answer == "12":
        correct_answer()
        print("Quiz completed! Well done.")
    else:
        wrong_answer()
        question_2()  # Repeat the question if the answer is incorrect


def correct_answer():
    print("Correct! Well done.")


def wrong_answer():
    print("That's incorrect. Try again.")


def start_quiz():
    print("Welcome to the quiz!")
    question_1()


# Start the quiz
if __name__ == "__main__":
    start_quiz()
```

Qn: Build a program for managing an inventory list by adding new items and updating

quantities. Create a Data Table variable to hold the inventory data (item names and

quantities).

```
import pandas as pd


# Initialize an empty inventory DataFrame
inventory = pd.DataFrame(columns=["Item", "Quantity"])


# Function to add a new item to the inventory
def add_item(item_name, quantity):
    global inventory
    # Check if item already exists
    if item_name in inventory["Item"].values:
        print(f"{item_name} already exists in the inventory. Use update_quantity
instead.")
    else:
        # Add new item
        new_item = pd.DataFrame([[item_name, quantity]], columns=["Item",
"Quantity"])
```

```
        inventory = pd.concat([inventory, new_item], ignore_index=True)
        print(f"Added {item_name} with quantity {quantity}.")

# Function to update the quantity of an existing item
def update_quantity(item_name, quantity):
    global inventory
    if item_name in inventory["Item"].values:
        inventory.loc[inventory["Item"] == item_name, "Quantity"] += quantity
        print(f"Updated {item_name} quantity by {quantity}. New quantity:
{inventory.loc[inventory['Item'] == item_name, 'Quantity'].values[0]}.")
    else:
        print(f"{item_name} not found in the inventory. Use add_item to add new
items.")

# Function to display the current inventory
def display_inventory():
    print("\nCurrent Inventory:")
    print(inventory.to_string(index=False))

# Sample usage
add_item("Apples", 50)
add_item("Oranges", 30)
display_inventory()

update_quantity("Apples", 20)
update_quantity("Bananas", 15)  # Example for an item not in the inventory
display_inventory()
```

Qn: Develop a program for scraping job titles, companies, and locations from a job portal.

Validate the output by opening the CSV file and checking the entries.

```
from selenium import webdriver

from selenium.webdriver.common.by import By

import pandas as pd

import time


# Set up the web driver (Make sure to have the ChromeDriver installed and in PATH)

driver = webdriver.Chrome()


# Navigate to a job portal's search results page (replace with actual job portal URL)

driver.get("https://www.example-job-portal.com/jobs")


# Allow the page to load

time.sleep(3)


# Lists to store the scraped data

job_titles = []

companies = []

locations = []


# Locate and scrape job listings

try:

  # Locate job elements on the page (modify based on actual site structure)

  job_listings = driver.find_elements(By.CLASS_NAME, "job-listing-class")  #
Replace with actual class name


  for job in job_listings:

    # Extract job title
```

```
        title = job.find_element(By.CLASS_NAME, "job-title-class").text  # Replace with
actual class name
        job_titles.append(title)

        # Extract company name
        company = job.find_element(By.CLASS_NAME, "company-name-class").text  #
Replace with actual class name
        companies.append(company)

        # Extract job location
        location = job.find_element(By.CLASS_NAME, "location-class").text  # Replace
with actual class name
        locations.append(location)

except Exception as e:
    print("Error during scraping:", e)

finally:
    # Close the browser
    driver.quit()

# Create a DataFrame from the scraped data
jobs_data = pd.DataFrame({
    "Job Title": job_titles,
    "Company": companies,
    "Location": locations
})

# Save the data to a CSV file
```

```
csv_filename = "job_listings.csv"

jobs_data.to_csv(csv_filename, index=False)

print(f"Job data saved to {csv_filename}")


# Validate by reading and displaying the CSV contents

print("\nValidating CSV output:")

try:

  data_check = pd.read_csv(csv_filename)

  print(data_check.to_string(index=False))  # Print the data for verification

except FileNotFoundError:

  print("CSV file not found.")
```

Qn: Develop a simple program for managing an order processing system with various

states based on user input. Create states for "Order Received," "Processing," "Shipped,"

and "Cancelled."

```
def order_received():

  print("Order received. Moving to 'Processing' state.")

  process_order()


def process_order():

  print("Order is now in 'Processing' state.")

  user_input = input("Enter 'ship' to ship the order or 'cancel' to cancel it: ").strip().lower()

  if user_input == 'ship':

    ship_order()

  elif user_input == 'cancel':

    cancel_order()

  else:
        print("Invalid input. Please enter 'ship' or 'cancel'.")
        process_order()  # Retry input


def ship_order():
    print("Order has been 'Shipped'. Thank you for your business!")


def cancel_order():
    print("Order has been 'Cancelled'. Sorry to see you cancel!")


def start_order_process():
    print("Starting order process.")
    order_received()


# Start the order processing system
if __name__ == "__main__":
    start_order_process()
```

Qn: Develop a program for collect feedback through a structured workflow. Define states

for "Feedback Requested," "Feedback Collected," and "Thank You."

```
def feedback_requested():
    print("Feedback Requested: Please provide your feedback.")
    feedback = input("Your feedback: ").strip()
    if feedback:
        feedback_collected(feedback)
    else:
        print("No feedback provided. Please try again.")
        feedback_requested()
```

```
def feedback_collected(feedback):
    print(f"Feedback Collected: Thank you for your feedback: '{feedback}'")
    thank_you()


def thank_you():
    print("Thank You: Your feedback has been received. We appreciate your input!")


def start_feedback_process():
    print("Starting feedback collection process.")
    feedback_requested()


# Start the feedback collection process
if __name__ == "__main__":
    start_feedback_process()
```

Qn: Build a python program to create a sequence in an RPA tool that collects user inputs

(like Full name, Phone number, Favorite color, Favorite hobby and Preferred vacation

destination) and then displays these using a message box.

```
import tkinter as tk
from tkinter import messagebox


def collect_user_inputs():
    # Create a window to collect user inputs
    window = tk.Tk()
    window.title("User Input Form")

    # Labels and entry fields for each input
    tk.Label(window, text="Full Name:").grid(row=0, column=0, padx=10, pady=5)
    full_name_entry = tk.Entry(window)
    full_name_entry.grid(row=0, column=1, padx=10, pady=5)

    tk.Label(window, text="Phone Number:").grid(row=1, column=0, padx=10, pady=5)
    phone_number_entry = tk.Entry(window)
    phone_number_entry.grid(row=1, column=1, padx=10, pady=5)

    tk.Label(window, text="Favorite Color:").grid(row=2, column=0, padx=10, pady=5)
    favorite_color_entry = tk.Entry(window)
    favorite_color_entry.grid(row=2, column=1, padx=10, pady=5)

    tk.Label(window, text="Favorite Hobby:").grid(row=3, column=0, padx=10, pady=5)
    favorite_hobby_entry = tk.Entry(window)
    favorite_hobby_entry.grid(row=3, column=1, padx=10, pady=5)

    tk.Label(window, text="Preferred Vacation Destination:").grid(row=4, column=0, padx=10, pady=5)
    vacation_destination_entry = tk.Entry(window)
    vacation_destination_entry.grid(row=4, column=1, padx=10, pady=5)

    # Function to display message box with user inputs
    def display_inputs():
        full_name = full_name_entry.get()
        phone_number = phone_number_entry.get()
        favorite_color = favorite_color_entry.get()
        favorite_hobby = favorite_hobby_entry.get()
        vacation_destination = vacation_destination_entry.get()
        message = f"Full Name: {full_name}\nPhone Number: {phone_number}\nFavorite Color: {favorite_color}\nFavorite Hobby: {favorite_hobby}\nPreferred Vacation Destination: {vacation_destination}"
        messagebox.showinfo("User Information", message)
        window.destroy()

    # Submit button to collect and display the data
    submit_button = tk.Button(window, text="Submit", command=display_inputs)
    submit_button.grid(row=5, columnspan=2, pady=20)

    window.mainloop()

# Start the process
collect_user_inputs()
```

Qn: Develop a flowchart that asks the user which section they want to visit (e.g., Home,

About, Contact).

```
def visit_section():
    print("Which section would you like to visit?")
    print("1. Home")
    print("2. About")
    print("3. Contact")

    choice = input("Enter the number of your choice: ").strip()

    if choice == '1':
        print("Welcome to the Home section!")
    elif choice == '2':
        print("Welcome to the About section!")
    elif choice == '3':
        print("Welcome to the Contact section!")
    else:
        print("Invalid choice, please try again.")
        visit_section()  # Retry if invalid input

# Start the process
if __name__ == "__main__":
    visit_section()
```

Qn: Build a program for Automating the process of reading files from a folder and

processing their contents. Use arguments to pass the directory path in and the processed

data out of the workflow.

```
import os
import argparse

def process_file(file_path):
    # Open and process the file contents
    with open(file_path, 'r') as file:
        content = file.read()
    # Example processing: return the number of words in the file
    word_count = len(content.split())
    return word_count

def process_files_in_directory(directory_path, output_file):
    # List all files in the directory
    files = os.listdir(directory_path)
    # Filter out only text files (you can adjust the file type)
    text_files = [f for f in files if f.endswith('.txt')]

    # Process each file and store the results
    results = []
    for file_name in text_files:
        file_path = os.path.join(directory_path, file_name)
        word_count = process_file(file_path)
        results.append(f"{file_name}: {word_count} words")

    # Write the processed data to the output file
    with open(output_file, 'w') as out_file:
        for result in results:
            out_file.write(result + '\n')
    print(f"Processed data written to {output_file}")

def main():
    # Set up argument parser
    parser = argparse.ArgumentParser(description='Process files in a directory.')
    parser.add_argument('directory', help='Directory path to process files from')
    parser.add_argument('output', help='Output file to store the processed data')

    # Parse arguments
    args = parser.parse_args()

    # Process the files and save the results
    process_files_in_directory(args.directory, args.output)
```

```
if __name__ == "__main__":
    main()
```