Course: Python

I. Final Project (code with comments)

```
import tkinter as tk
from tkinter import ttk, scrolledtext, messagebox
import requests
from bs4 import BeautifulSoup
import threading
import pandas as pd
import time


class MovieScraperApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Movie Suggestion App")
        self.root.geometry("600x500")

        self.style = ttk.Style()
        self.style.configure('TLabel', font=('Arial', 12))
        self.style.configure('TButton', font=('Arial', 12))

        self.create_widgets()

    def create_widgets(self):
        self.genre_label = ttk.Label(self.root, text="Select a movie genre:")
        self.genre_label.pack(pady=10)

        self.genres = ['Action', 'Adventure', 'Comedy', 'Drama', 'Fantasy', 'Horror', 'Mystery', 'Romance', 'Sci-
Fi', 'Thriller']
        self.genre_combobox = ttk.Combobox(self.root, values=self.genres, state='readonly', width=30)
```

```
    self.genre_combobox.pack()

    self.top_label = ttk.Label(self.root, text="Select number of top movies:")
    self.top_label.pack(pady=10)

    self.top_combobox = ttk.Combobox(self.root, values=[3, 5, 10], state='readonly', width=10)
    self.top_combobox.pack()

    self.fetch_button = ttk.Button(self.root, text="Fetch Movies", command=self.fetch_and_display)
    self.fetch_button.pack(pady=20)

    self.clear_button = ttk.Button(self.root, text="Clear Screen", command=self.clear_screen)
    self.clear_button.pack(pady=10)

    self.results_text = scrolledtext.ScrolledText(self.root, height=15, width=70, wrap=tk.WORD)
    self.results_text.pack(pady=20)

def fetch_and_display(self):
    genre = self.genre_combobox.get()
    if not genre:
        messagebox.showerror("Error", "Please select a genre.")
        return

    top_number = self.top_combobox.get()
    if not top_number:
        messagebox.showerror("Error", "Please select number of top movies.")
        return

    self.results_text.delete('1.0', tk.END)
    self.results_text.insert(tk.END, f"Fetching movies for {genre} genre..\n")
```

```
        self.root.update()  # Update GUI to show fetching message

        # Call method to scrape and display movies using a separate thread
        threading.Thread(target=self.fetch_movies, args=(genre, int(top_number))).start()

    def fetch_movies(self, genre, top_number):
        url = f'https://www.rottentomatoes.com/browse/movies_in_theaters/genres:{genre.lower()}'
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/91.0.4472.124 Safari/537.36'
        }

        try:
            response = requests.get(url, headers=headers)
            response.raise_for_status()  # Check if the request was successful

            soup = BeautifulSoup(response.content, 'html.parser')
            movies = []

            movie_tags = soup.select('span.p--small[data-qa="discovery-media-list-item-title"]')
            for tag in movie_tags:
                movie_name = tag.get_text(strip=True)
                movies.append(movie_name)

            # Export movies to an Excel file
            self.export_to_excel(movies, genre)


            # Clear results text and show movies
            self.results_text.delete('1.0', tk.END)
```

```
        self.results_text.insert(tk.END, f"Catalog Received for {genre} genre:\n\n")
        self.results_text.insert(tk.END, f"Top {top_number} {genre} movies:\n\n")
        self.root.update()  # Update GUI to show catalog received message

        # Read from Excel file and display top movies
        self.read_from_excel_and_display(genre, top_number)

    except requests.exceptions.RequestException as e:
        messagebox.showerror("Network Error", f"Failed to fetch movies: {e}")
    except Exception as e:
        messagebox.showerror("Error", f"An error occurred: {e}")

def export_to_excel(self, movies, genre):
    """Export the list of movies to an Excel file"""
    df = pd.DataFrame(movies, columns=['Movie Name'])
    excel_filename = f"scraped_raw_{genre.lower()}_movies.xlsx"
    df.to_excel(excel_filename, index=False)
    self.results_text.insert(tk.END, f"Movies exported to {excel_filename}\n")
    self.root.update()

def read_from_excel_and_display(self, genre, top_number):
    """Read movies from the Excel file and display the top movies"""
    excel_filename = f"scraped_raw_{genre.lower()}_movies.xlsx"
    df = pd.read_excel(excel_filename)
    top_movies = df['Movie Name'].head(top_number)

    # Print each movie with a delay
    for idx, movie in enumerate(top_movies, start=1):
        self.results_text.insert(tk.END, f"{idx}. {movie}\n")
        self.root.update()  # Update the GUI to show each movie
```

```
        time.sleep(0.5)  # Add a delay between displaying each movie

    # Add final message after displaying all movies
    self.results_text.insert(tk.END, "\nPlease enjoy the movies!\n")

  def clear_screen(self):
    self.genre_combobox.set('')  # Clear genre selection
    self.top_combobox.set('')  # Clear top number selection
    self.results_text.delete('1.0', tk.END)  # Clear text area

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

II. Explanation (Step by step)

Movie_Suggestion_App_1.0.py > MovieScraperApp

```
import tkinter as tk
from tkinter import ttk, scrolledtext, messagebox
import requests
from bs4 import BeautifulSoup
import threading
import pandas as pd
import time

```

- **Libraries Import**: This block imports necessary libraries for GUI creation (tkinter), web scraping (requests, BeautifulSoup), threading, data handling (pandas), and introducing delays (time).

```python
class MovieScraperApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Movie Suggestion App")
        self.root.geometry("600x500")

        self.style = ttk.Style()
        self.style.configure('TLabel', font=('Arial', 12))
        self.style.configure('TButton', font=('Arial', 12))

        self.create_widgets()

```

- **Initializing a class**: The class named MovieScraperApp initializes the GUI window, sets its title, dimensions, and styles using tkinter and ttk.

```python
    def create_widgets(self):
        self.genre_label = ttk.Label(self.root, text="Select a movie genre:")
        self.genre_label.pack(pady=10)

        self.genres = ['Action', 'Adventure', 'Comedy', 'Drama', 'Fantasy', 'Horror', 'Mystery', 'Romance'
        self.genre_combobox = ttk.Combobox(self.root, values=self.genres, state='readonly', width=30)
        self.genre_combobox.pack()

        self.top_label = ttk.Label(self.root, text="Select number of top movies:")
        self.top_label.pack(pady=10)

        self.top_combobox = ttk.Combobox(self.root, values=[3, 5, 10], state='readonly', width=10)
        self.top_combobox.pack()

        self.fetch_button = ttk.Button(self.root, text="Fetch Movies", command=self.fetch_and_display)
        self.fetch_button.pack(pady=20)

        self.clear_button = ttk.Button(self.root, text="Clear Screen", command=self.clear_screen)
        self.clear_button.pack(pady=10)

        self.results_text = scrolledtext.ScrolledText(self.root, height=15, width=70, wrap=tk.WORD)
        self.results_text.pack(pady=20)

```

- **Creating Widgets**: This function creates and packs the widgets (labels, combo boxes, buttons, and text area) into the window. It includes selection options for genres and the number of movies to fetch, as well as buttons to fetch movies and clear the screen.

```python
    def fetch_and_display(self):
        genre = self.genre_combobox.get()
        if not genre:
            messagebox.showerror("Error", "Please select a genre.")
            return

        top_number = self.top_combobox.get()
        if not top_number:
            messagebox.showerror("Error", "Please select number of top movies.")
            return

        self.results_text.delete('1.0', tk.END)
        self.results_text.insert(tk.END, f"Fetching movies for {genre} genre..\n")
        self.root.update()  # Update GUI to show fetching message

        # Call method to scrape and display movies using a separate thread
        threading.Thread(target=self.fetch_movies, args=(genre, int(top_number))).start()

```

- **Fetching and Displaying Movies**: This function gets the selected genre and number of movies, displays a fetching message, and starts a new thread to fetch movies to keep the GUI responsive.

```python
    def fetch_movies(self, genre, top_number):
        url = f'https://www.rottentomatoes.com/browse/movies_in_theaters/genres:{genre.lower()}'
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/5
        }

        try:
            response = requests.get(url, headers=headers)
            response.raise_for_status()  # Check if the request was successful

            soup = BeautifulSoup(response.content, 'html.parser')
            movies = []

            movie_tags = soup.select('span.p--small[data-qa="discovery-media-list-item-title"]')
            for tag in movie_tags:
                movie_name = tag.get_text(strip=True)
                movies.append(movie_name)

            # Export movies to an Excel file
            self.export_to_excel(movies, genre)

            # Clear results text and show movies
            self.results_text.delete('1.0', tk.END)
            self.results_text.insert(tk.END, f"Catalog Received for {genre} genre:\n\n")
            self.results_text.insert(tk.END, f"Top {top_number} {genre} movies:\n\n")
            self.root.update()  # Update GUI to show catalog received message

            # Read from Excel file and display top movies
            self.read_from_excel_and_display(genre, top_number)

        except requests.exceptions.RequestException as e:
            messagebox.showerror("Network Error", f"Failed to fetch movies: {e}")
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {e}")
```

- **Fetching Movies**: This function builds the URL for scraping, makes the HTTP request, parses the HTML response to extract movie titles, and exports the data to an Excel file. It handles errors and updates the GUI.

```python
    def export_to_excel(self, movies, genre):
        """Export the list of movies to an Excel file"""
        df = pd.DataFrame(movies, columns=['Movie Name'])
        excel_filename = f"scraped_raw_{genre.lower()}_movies.xlsx"
        df.to_excel(excel_filename, index=False)
        self.results_text.insert(tk.END, f"Movies exported to {excel_filename}\n")
        self.root.update()

```

- **Exporting to Excel**: This function takes a list of movies and exports it to an Excel file named based on the genre.

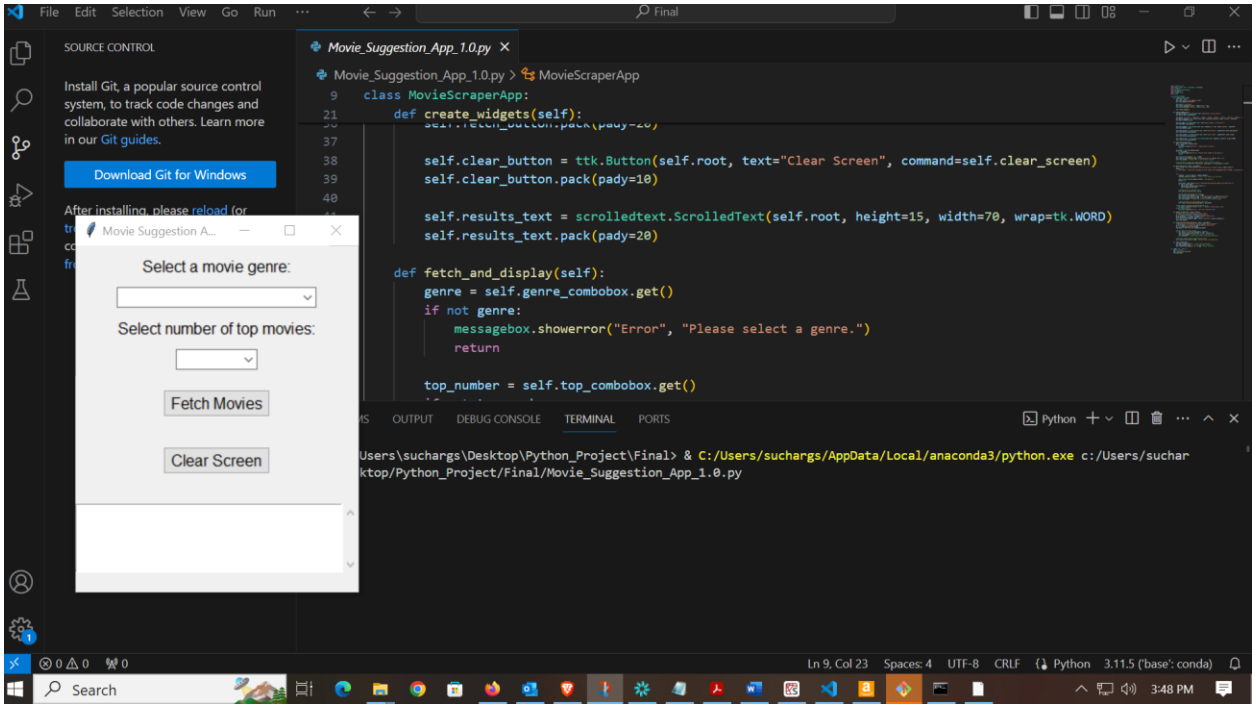
```python
    def read_from_excel_and_display(self, genre, top_number):
        """Read movies from the Excel file and display the top movies"""
        excel_filename = f"scraped_raw_{genre.lower()}_movies.xlsx"
        df = pd.read_excel(excel_filename)
        top_movies = df['Movie Name'].head(top_number)

        # Print each movie with a delay
        for idx, movie in enumerate(top_movies, start=1):
            self.results_text.insert(tk.END, f"{idx}. {movie}\n")
            self.root.update()  # Update the GUI to show each movie
            time.sleep(0.5)  # Add a delay between displaying each movie

        # Add final message after displaying all movies
        self.results_text.insert(tk.END, "\nPlease enjoy the movies!\n")

    def clear_screen(self):
        self.genre_combobox.set('')  # Clear genre selection
        self.top_combobox.set('')  # Clear top number selection
        self.results_text.delete('1.0', tk.END)  # Clear text area

```

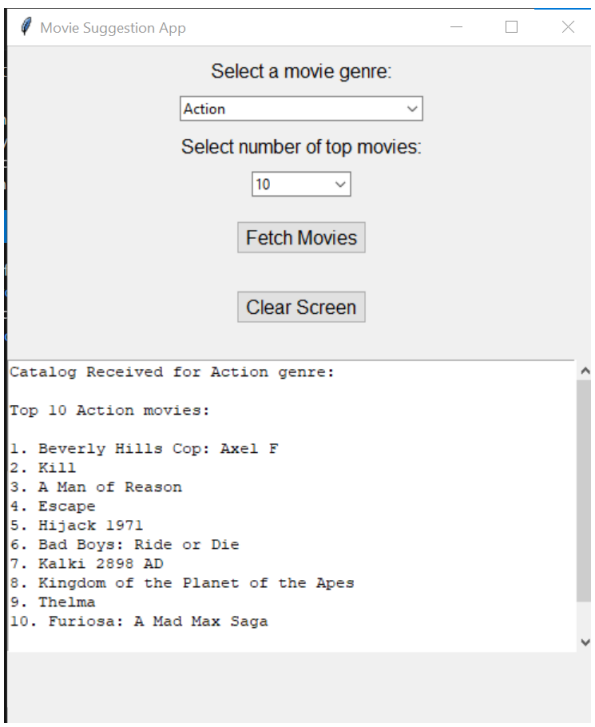
- **Reading from Excel and Displaying**: This function reads the movie list from the Excel file using pandas, extracts the top movies and displays them in the GUI.

```python
    def clear_screen(self):
        self.genre_combobox.set('')  # Clear genre selection
        self.top_combobox.set('')  # Clear top number selection
        self.results_text.delete('1.0', tk.END)  # Clear text area

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

```

- **Clearing the Screen**: This function resets the genre and top number selections and clears the text area in the GUI and then creates the main Tkinter window, initializes the MovieScraperApp class, and starts the Tkinter main loop.

III. Output


File Edit Selection View Go Run
SOURCE CONTROL
Install Git, a popular source control system, to track code changes and collaborate with others. Learn more in our Git guides.
Download Git for Windows
Movie_Suggestion_App_1.0.py
Movie_Suggestion_App_1.0.py > MovieScraperApp
class MovieScraperApp:
def create_widgets(self):
self.clear_button = ttk.Button(self.root, text="Clear Screen", command=self.clear_screen)
self.clear_button.pack(pady=10)
self.results_text = scrolledtext.ScrolledText(self.root, height=15, width=70, wrap=tk.WORD)
self.results_text.pack(pady=20)
def fetch_and_display(self):
genre = self.genre_combobox.get()
if not genre:
messagebox.showerror("Error", "Please select a genre.")
return
top_number = self.top_combobox.get()
Movie Suggestion A...
Select a movie genre:
Select number of top movies:
Fetch Movies
Clear Screen
OUTPUT DEBUG CONSOLE TERMINAL PORTS
Users\suchargs\Desktop\Python_Project\Final> & C:/Users/suchargs/AppData/Local/anaconda3/python.exe c:/Users/suchar
ktop/Python_Project/Final/Movie_Suggestion_App_1.0.py
Ln 9, Col 23 Spaces: 4 UTF-8 CRLF Python 3.11.5 ('base': conda)
Search
3:48 PM


Movie Suggestion App
Select a movie genre:
Action
Select number of top movies:
10
Fetch Movies
Clear Screen
Catalog Received for Action genre:
Top 10 Action movies:
1. Beverly Hills Cop: Axel F
2. Kill
3. A Man of Reason
4. Escape
5. Hijack 1971
6. Bad Boys: Ride or Die
7. Kalki 2898 AD
8. Kingdom of the Planet of the Apes
9. Thelma
10. Furiosa: A Mad Max Saga