50 Days of Code - The Complete Python Pro Bootcamp

This comprehensive Python bootcamp offers an in-depth journey into Python programming, starting from beginner-level concepts and advancing to professional-grade development. Over the course of 50 days, learners are guided through hands-on projects and real-world applications, ensuring mastery of both foundational and advanced Python topics.

Key areas covered include:

- **Core Python:** Learning Python 3, including scripting, automation, and object-oriented programming.
- **Game Development:** Build interactive games using libraries like Turtle.
- **Web Scraping:** Extract data from websites with tools like Beautiful Soup and Selenium.
- **Data Science:** Dive into data analysis using Pandas, NumPy, Matplotlib, Seaborn, and more.
- **Web Development:** Develop both front-end and back-end applications using Flask, REST APIs, and databases like SQL, SQLite, and PostgreSQL.
- **Automation & GUI Development:** Automate tasks and build GUI desktop applications using Tkinter.
- **Deployment & Version Control:** Learn to deploy web apps and manage code with Git, GitHub, and Heroku.

The course emphasizes project-based learning, culminating in a portfolio of 50 Python projects, ranging from games to web apps, data analysis tools, and automation scripts. It provides a solid foundation for anyone looking to become a professional Python developer or enhance their programming skill set.

Starting the 50 Days of Code Challenge

Embarking on the 50 Days of Code challenge has been an exciting personal journey to enhance my Python skills. With a commitment to coding for at least one hour a day, I'm focusing on building practical projects that reinforce the core concepts of programming. Each day introduces new challenges, helping me to progressively improve my problem-solving abilities and deepen my understanding of Python. This journey is not just about writing code, but about building a strong foundation that can be applied to real-world applications.

Day 26 : NATO Alphabet Project :

This project involves building a simple Python program that translates a user's input into the corresponding NATO phonetic alphabet code. The goal is to create a user-friendly tool where users can input any word, and the program will return the corresponding phonetic alphabet representation for each letter of the word.

Learning Objectives and Key Concepts:

Data Manipulation with Pandas: The project starts by loading data from a CSV file that contains the NATO phonetic alphabet. This file includes two columns: one for the letters (A-Z) and another for their corresponding phonetic code words (e.g., "A" for "Alfa", "B" for "Bravo"). Using the Pandas library, we read this file and transform the data into a dictionary where each letter is a key, and its corresponding code word is the value.

Dictionary Comprehension: The NATO phonetic alphabet dictionary is created using a Python dictionary comprehension. This efficient method of iterating through the rows of a Pandas DataFrame using iterrows() allows us to quickly build the dictionary from the CSV file data.

User Input Handling: The program prompts the user to input a word, which is then converted to uppercase to ensure consistency with the keys in the dictionary. This step is important since the phonetic alphabet is case-insensitive, but the program's dictionary keys are in uppercase.

List Comprehension for Phonetic Translation: After receiving the user input, the program uses list comprehension to generate a list of phonetic code words for each letter in the input word. This step demonstrates the power of Python's list comprehensions for creating clean, concise code.

Dictionary and List Comprehensions: This project deepened my understanding of using comprehensions in Python to create dictionaries and lists from iterable data sources.

Day 27 : Miles to Kilometers Converter :

This project involves creating a simple graphical user interface (GUI) application that converts distances from miles to kilometers. The application is built using the Python Tkinter library, allowing for easy interaction with users through a window where they can input values and receive instant feedback.

Learning Objectives and Key Concepts:

GUI Design with Tkinter: The primary objective of this project was to get hands-on experience with building a GUI application using Tkinter. By using basic widgets like labels, buttons, and input fields, I was able to construct an intuitive user interface that enables users to perform quick conversions.

Widget Placement with Grid Layout: Tkinter's grid() method was used to organize the elements (widgets) in a grid layout, which allowed for a clean and structured arrangement of components. This project reinforced the importance of layout management for building visually appealing interfaces.

User Input and Output Handling: The Entry widget was used to take user input (the number of miles), and the output (in kilometers) was displayed using a Label widget. The program processes the user input, performs the miles-to-kilometers conversion, and updates the result dynamically within the interface.

Basic Conversion Logic:
The conversion formula for miles to kilometers is simple but important:
1 mile = 1.609 kilometers. This project helped reinforce basic mathematical operations in Python while demonstrating how to integrate such logic into a real-world application.

Button Click Event Handling: I implemented an action that gets triggered when the user clicks the "Calculate" button. This click event is linked to a function that retrieves the input, processes it, and updates the output label with the converted value.

Day 28: Pomodoro Timer :

This project involves building a Pomodoro Timer using the Tkinter library in Python. The Pomodoro technique is a time management method that breaks work into intervals, typically 25 minutes in length, separated by short breaks. The goal of this project was to create a visual timer that helps users follow the Pomodoro method with a countdown mechanism and notifications for work and break sessions.

Learning Objectives and Key Concepts:

GUI Development with Tkinter: This project reinforced my knowledge of Tkinter, the built-in Python library for creating graphical user interfaces. I learned how to create and manipulate various widgets such as buttons, labels, and canvases, as well as how to apply custom styles (colors, fonts, etc.) to make the interface user-friendly.

State Management and Event Handling: A key feature of this project is state management, as the timer alternates between work and break periods. I used event handling to implement start and reset functions for the timer, updating the label text and colors dynamically based on the current state (work, short break, or long break).

Countdown Mechanism: The core of the application is the countdown function, which recursively calls itself every second until the timer reaches zero. This process helps simulate the Pomodoro technique by triggering work and break intervals in sequence.

Repetition Tracking: The project uses a repetition counter to keep track of the intervals (referred to as rep). Every set of work and break sessions is marked by a check mark, which visually shows how many Pomodoro cycles the user has completed. This required careful logic to ensure that after every two sessions, a check mark was displayed.

Time Management Functions: I utilized mathematical functions like math.floor() to calculate minutes and seconds from the total number of seconds in a given interval, ensuring that the timer displays in the correct format.

This project was a great practical exercise in combining time management techniques with GUI development, while also demonstrating how to handle real-time updates in Python applications.

Day 29 : Password Manager :

In this project, I developed a Password Manager application using Python's Tkinter library. The purpose of the project was to create a user-friendly interface where users can generate, store, and manage passwords securely. The password manager allows users to input a website, their email/username, and a generated password, which can be stored in a local file.

Learning Objectives and Key Concepts:

Password Generation: A random password generator was implemented, using a combination of letters, numbers, and special symbols. The random module was utilized to create strong passwords with customizable length and character composition. This feature ensures security by generating unpredictable passwords.

GUI Development with Tkinter: Tkinter was used extensively to build a graphical user interface (GUI) that interacts with the user. Widgets such as Label, Entry, and Button were utilized to create a clean and functional interface. Through this, I gained experience in handling layout and configuring UI elements to make the application user-friendly.

Entry Validation: Input validation was included to ensure the fields (website, email/username, and password) are correctly filled before saving. Appropriate error handling was implemented using message boxes (messagebox.showwarning) to prompt the user to fill all required fields or to notify them of inputs that are too short.

Clipboard Functionality: The generated password is automatically copied to the clipboard using the pyperclip library. This eliminates the need for users to manually copy passwords, enhancing convenience.

File Handling: The user data, including the website, email/username, and password, is saved into a local CSV file (passwords_dict.csv). This ensures that the password records can be stored and retrieved later. File operations such as opening, appending, and writing data were managed efficiently using Python’s file handling mechanisms.

Confirmation Dialogs: Before saving any data, the application displays a confirmation dialog (messagebox.askokcancel) to ensure that the entered information is correct. This adds a layer of security, allowing users to double-check their inputs before storing them permanently.

Day 30 : Password Manager Project

This project is a simple Password Manager application built using Python's tkinter library for the graphical user interface (GUI). The program allows users to manage and securely store passwords for different websites. I implemented several key features that enhance both the usability and functionality of the application.

Key Features:

Password Generation:
The application includes a built-in password generator. When a user clicks the "Generate Password" button, a random password is created using an external password_generator module. The generated password is automatically inserted into the password field and copied to the clipboard for easy use.

Password Storage:
Passwords, along with the associated website and email/username, are stored in a passwords.json file. This allows for easy retrieval and reuse later. If the file doesn't exist, it is created, and new entries are appended to the existing data.

Search Functionality:
Users can search for passwords by entering the website name. The application retrieves the relevant email/username and password from the passwords.json file and displays it in a messagebox. If the website isn't found, an appropriate warning is displayed.

Form Validation:
Before adding any new entries, the application validates the input to ensure that no fields are left empty and that the website, email, and password meet a minimum length requirement. This improves both user experience and data integrity.

Data Persistence:
Passwords are saved to a JSON file, ensuring that data is persisted even after closing the application. The file is updated each time a new entry is added.

Error Handling: implemented exception handling, particularly when reading and writing to the JSON file. This ensures that the program remains robust in cases where the file may be missing or corrupted.

Day 31 : Flashcard Application :

This project is a Flashcard Application designed to help users learn Arabic vocabulary through an interactive graphical user interface (GUI) built with Python's tkinter library. The application randomly presents Arabic words alongside their English translations, allowing users to test their knowledge and reinforce their learning.

Key Features:

Dynamic Vocabulary Learning:
Users can learn Arabic words from a CSV file. If the user has previously learned words, the application will use this data to continue their learning journey. If the file is not found, the app loads a default set of Arabic words.

Flashcard Interface:
The application displays flashcards with Arabic words. Users can see the Arabic word and, after a set duration, the English translation. This flip feature helps in retaining vocabulary by associating the foreign language with its translation.

User Interaction:
Users can indicate their understanding of the word by pressing buttons for "Correct" or "Incorrect." The application tracks user performance, maintaining a score that updates dynamically as users interact with the flashcards.

Progress Tracking:
The application keeps track of the user's score and the number of attempts, providing feedback on their learning progress.

Data Persistence:
Users’ progress is saved into a CSV file, allowing them to continue from where they left off during their next session.

Day 32 :Birthday Reminder Application :

This project is a Birthday Reminder Application that allows users to store, manage, and send birthday wishes via email. The application combines data handling and email functionality to create a personalized greeting experience for friends and family.

Key Features:

Data Storage:
Users can input names, email addresses, and birthdays, which are stored in a CSV file. This feature allows easy management of birthday information.

Birthday Checking:
The application checks the current date against the stored birthdays. If it finds any matches, it collects the names and email addresses of those whose birthdays fall on that day.

Personalized Letters:
The application randomly selects a birthday letter template from a predefined set. Each letter template includes a placeholder for the recipient's name, which is dynamically replaced with the actual name of the person whose birthday it is.

Email Sending:
Using the Simple Mail Transfer Protocol (SMTP), the application sends personalized birthday greetings to the corresponding email addresses. This functionality ensures that users can wish their loved ones directly through the application.

Date and Time Manipulation: The project allowed me to work with the datetime module to handle and compare dates, ensuring that birthday notifications are accurate and timely.

Email Communication via SMTP: experience in using the smtplib library to send emails programmatically, learning how to manage email authentication and handle potential errors in the process.

Day 33 : ISS Tracker and Notification System:

This project is an ISS Tracker and Notification System, designed to notify the user via email when the International Space Station (ISS) is overhead and it's dark outside at their location so they can see it. The application combines real-time data from APIs with email alerts, providing a practical and interesting project centered around space observation.

Key Features:

ISS Position Tracking:
The application uses the Open Notify API to fetch the current position of the ISS in real time. It then checks if the ISS is within a 5-degree radius of the user's latitude and longitude.

Daylight Checking:
The Sunrise-Sunset API is used to determine whether it is currently dark at the user's location. The application compares the current time with the sunrise and sunset times to ensure the ISS is only visible when it's dark.

Automated Email Notifications:
When the ISS is overhead and it's dark, the application sends an email to notify the user to look up and spot the ISS. It uses SMTP for email sending, and the email content is personalized to provide real-time alerts.

Efficient Use of Time Intervals:
The program checks the ISS position and daylight status every 60 seconds using the time.sleep() function, ensuring that the system remains efficient and avoids unnecessary API calls.

API Interaction: This project deepened my understanding of how to retrieve and manipulate data from web APIs using the requests library. I learned how to handle response data, such as JSON objects, and how to pass parameters effectively.

Geospatial Calculations: how to compare geographical coordinates to determine proximity, using latitude and longitude values to calculate if the ISS is overhead.

Day 34 : Quiz Application with GUI :

This project is a Quiz Application built with Python, utilizing the Tkinter library for a user-friendly graphical interface. The application integrates with the Open Trivia Database API to fetch trivia questions, making each quiz dynamic and unique. The project demonstrates the integration of API-driven content with GUI components, providing an interactive quiz experience.

Project Overview

Class Structure and Quiz Logic:

- The application is structured around three main classes: Question, QuizBrain, and QuizInterface.
- Question stores each question and its answer.
- QuizBrain handles the quiz flow, question tracking, and scoring, as well as checking if the user's response is correct.
- QuizInterface builds the graphical interface, presenting questions to the user, managing the score display, and updating visuals based on answers.

Fetching Questions from an API:
Trivia questions are fetched from the Open Trivia Database API in real time, based on parameters such as difficulty level and question type. The API response is processed to create question instances that are loaded into the quiz.

GUI with Tkinter:
The interface is designed with Tkinter, featuring a label to display the score, a canvas for the questions, and buttons for user interaction. True/False buttons allow users to submit answers, and the canvas changes color to indicate whether an answer was correct or incorrect.

Dynamic Question Loading and Scoring:
The application dynamically loads each question, updating the display and score based on user responses. Once all questions are answered, the app disables the answer buttons and displays a completion message.

Day 35 : Weather Alert Notification :

This project is a Weather Alert Notification System that uses the OpenWeatherMap API to forecast upcoming weather and sends an SMS alert via Twilio if rain is expected. The application demonstrates the integration of a weather data API with SMS notification services, providing real-time updates to help users prepare for rain.

Project Overview

API Integration for Weather Data:
The app pulls weather data using the OpenWeatherMap API. Forecast data for the specified location is requested and processed to check for rain conditions in the upcoming hours.

Automated Notification via Twilio:
When rain is detected in the forecast, the app uses Twilio's API to send an SMS alert. This alert serves as a proactive reminder to carry an umbrella. The Twilio credentials and weather API key are securely accessed from environment variables to protect sensitive information.

Configuration and Parameters:
Location parameters (latitude and longitude) are customizable, allowing flexibility to monitor weather for different locations.
Forecast count is set to analyze the next few hours, keeping alerts relevant and timely.

Environment Variable Management: Sensitive data such as API keys and authorization tokens are managed via environment variables, which enhances security and prevents hardcoding sensitive information.

Day 36 : Stock Alert with News Notification:

This project is a Stock Alert System that monitors the stock price of a specific company (in this case, Tesla Inc) and sends an SMS notification if the stock price experiences a significant change (more than 5%) from the previous day. The notification includes the percentage change and the latest news headlines related to the company.

Project Overview:

Stock Data Retrieval:
The system uses the Alpha Vantage API to fetch daily stock data for Tesla Inc (or any specified stock symbol).
It calculates the percentage change in the stock price compared to the previous day's closing price.

News Retrieval:
The News API is used to gather the latest news articles related to Tesla Inc. The system fetches the top 3 most recent articles.

SMS Notification:
When a significant change (greater than 5%) is detected in the stock price, an SMS notification is sent via Twilio API. The message contains:
The percentage change in stock price (up or down).
The headline and brief description of the latest news articles.

Day 37 : Habit Tracking with Pixela :

This project is a habit tracking system developed using the Pixela API, allowing users to visually track their habits, in this case, coding hours, over time. By interacting with Pixela's API, users can create a personal graph to log daily habits, update the data, and delete records if necessary.

Project Overview

User Creation:
The script initially sets up a new user on Pixela using a unique token and username. This account serves as the basis for storing and viewing habit data.

Graph Setup:
After user setup, a custom graph is created with parameters such as graph ID, name, unit of measurement (hours, in this case), data type, and color for visual representation. This graph will display the habit tracking data over time.

Daily Habit Tracking:
Each day's activity can be logged as a "pixel" on the graph, with the quantity representing the hours spent coding that day.
Data is stored using a formatted date string (YYYYMMDD) to track each day's entry accurately.

Data Update and Deletion:
The system allows for updating a specific day's data, useful for correcting or adjusting logged hours.
Additionally, it supports deleting records if a habit entry needs to be removed.

Day 38 : Workout Tracking Integration :

This project allows users to track their workouts and log the data into a Google Sheet using Nutritionix API for exercise data and Sheety API for interacting with Google Sheets. The system takes user input about their exercise type and duration, retrieves nutritional data (like calories burned), and logs the details into a personal Google Sheet.

Project Overview

User Input:
The user is prompted to input the type of exercise and its duration in minutes. This input is used to query the Nutritionix API, which returns detailed information on the exercise performed, such as calories burned.

Exercise Data:
The program uses the Nutritionix API to send the user's exercise query, including gender, weight, height, and age. It returns the duration of the exercise and the calories burned, which are extracted from the response.

Log to Google Sheets:
Using the Sheety API, the workout details are automatically logged to a Google Sheet. Each entry includes the date, time, type of exercise, duration, and calories burned.

Error Handling:
The program checks for the success of adding data to the Google Sheet, providing feedback on whether the data was successfully added or if the request failed.

Key Learning:

API Integration: learning how to integrate two different APIs — Nutritionix for exercise data and Sheety for interacting with Google Sheets. This demonstrated how to pass data between different systems seamlessly.

Data Extraction and Formatting: By extracting key details from API responses (such as calories and duration) and formatting them for logging, I gained hands-on experience in working with JSON data and formatting strings (e.g., current date and time).

Automated Data Logging: Automating the process of tracking workouts by interacting with Google Sheets helped streamline the workout logging process, making it easier to track progress over time.

Day 39 Flight Deals Tracking System :

Overview

This project is a comprehensive flight deals tracking system that utilizes APIs for fetching flight details, managing data, and sending notifications. It automates the process of checking for the cheapest flights and notifies subscribed users of new deals.

Features

Flight Data Management:
The project uses a Google Sheet to store cities and track flight prices.
It leverages the Sheety API to manage the Google Sheet, including retrieving and updating city data, IATA codes, and flight prices.

IATA Code Lookup:
Missing IATA codes for cities are identified using the Amadeus API. The codes are then updated in the Google Sheet to ensure accurate flight searches.

Flight Search:
Flights are searched using the Amadeus API, focusing on round-trip, non-stop flights.
The search is conducted for a date range starting from the next day and extending up to six months.
The system retrieves details such as flight price, departure and arrival airports, and timings.

Notification System:
Users who subscribe via a Google Form are notified via email when a flight deal is found.
Emails include personalized messages with details about the flight, including price, departure, and arrival information.

Automation:
Data management, flight searches, and notifications are automated, ensuring seamless operation with minimal manual intervention.

Key Components

Data Manager:
Retrieves city data and flight prices from the Google Sheet.
Updates missing IATA codes and flight prices.

Flight Search:
Authenticates with the Amadeus API.
Fetches IATA codes for cities and searches for flights.
Finds the cheapest available flights for a given city.

Flight Data:
Orchestrates the interaction between the data manager and flight search modules.
Ensures that IATA codes and flight data are correctly updated.

Notification Manager:
Sends email notifications to users when a new flight deal is available.

Users Data:
Manages user subscription data from the Google Sheet.
Sends personalized emails to subscribed users about flight deals.

Day 40: personal project:

On Day 40, I worked on a personal project aimed at making my life a little easier: moving my YouTube Music playlist to Spotify. The idea was born from the frustration of manually searching and adding every single song from my playlist to Spotify. After some research, I discovered that I could use Python to automate the process.

I started by using BeautifulSoup to scrape my YouTube Music recap page for song titles. Then, I used the Spotipy library to search for these songs on Spotify and add them to a new playlist. The result was a much more efficient way of transferring my music without needing to search for each track individually. This project not only saved me time but also demonstrated the power of Python in automating tasks and solving real-world problems.

Day 41: Introduction to HTML:

Summary

On Day 41, the focus was on learning the basics of HTML to structure web content. The project involved creating a simple webpage that listed the top three favorite movies, each with a brief description. The exercise emphasized understanding the fundamental elements of HTML, such as headings, paragraphs, and horizontal rules, while organizing content in a visually appealing and structured format.

Key Learnings

HTML Basics: Mastered the use of headings (<h1> to <h3>), paragraphs (<p>), and horizontal rules (<hr />) for structuring a webpage.
Content Organization: Learned how to format and prioritize content hierarchically using HTML tags.
Project Application: Applied these concepts to create a personalized project, showcasing favorite movies with meaningful descriptions.

Day 42: Foundations of HTML - Birthday Invitation Project

Key Learnings:

Structuring a Web Page: Used headings (<h1>, <h2>, <h3>) to create a clear hierarchy of information on the webpage.
Embedding Images and Links: Implemented an image with an alt attribute for accessibility and linked to an external resource (Google Maps) using an <a> tag.
Unordered and ordered Lists: Organized items (e.g., balloons, cake) into a list using <ul> and <li> tags.
Styling Enhancements: Added horizontal rules (<hr>) to visually separate sections for better readability.

Project Summary:

The project focused on building a birthday invitation webpage featuring event details, a themed image, a checklist of items to bring, and a map link for the venue. It emphasized structuring content effectively while practicing fundamental HTML tags and attributes.

Day 43:HTML Foundations - Spanish Vocabulary Project:

Key Learnings:

HTML Structure and Content: Used a <h1> heading to title the page and <h2> headings to display various color names in Spanish, associating each color with an image.
Class and ID Attributes: Employed both class and ID attributes for styling and to uniquely identify specific elements (e.g., for color names and images).
Image Embedding: Used the <img> tag to embed images corresponding to each color. Each image had an alt attribute for accessibility, describing the color.
CSS Styling: Styled the page with CSS by adjusting the font size, color, and layout of the headings, and customized each color name using IDs to match the actual colors. The images were uniformly resized for consistency.

Project Summary:

This project focused on building a simple Spanish vocabulary page featuring common color names. The page displayed the color names in both Spanish and English, along with corresponding images for visual reference. CSS was used to style the text and images, enhancing the user experience and providing a visual learning tool for vocabulary acquisition.

Day 44: HTML Foundations - Motivation Meme Project:

Key Learnings:

Page Layout with CSS: Centered content using a container (meme-container) styled to occupy half the width of the screen and aligned it in the middle using margin adjustments.

Typography Customization: Applied a custom font, Libre Baskerville, using Google Fonts. Adjusted the typography for headers and paragraphs to emphasize structure and readability.

Image Integration and Styling: Incorporated an image into the design, styled with a uniform border and responsive width to fit within the container.

Theme and Aesthetic Design: Created a visually appealing theme with a black background, white text, and centered content for a polished, professional appearance.

Project Summary:

This project involved creating a simple and visually appealing meme page that displayed a motivational image with a caption and subtext. The use of external fonts, careful styling of the text and image, and a clean design showcased the integration of HTML and CSS for creating aesthetic web pages.

Day 45: Web Scraping with BeautifulSoup:

Key Learnings:

BeautifulSoup for Web Scraping: Learned how to use BeautifulSoup to parse and navigate HTML content. It allows extracting data from web pages by identifying elements like tags and classes.

Sending HTTP Requests: Used the requests library to fetch HTML content from a URL. This is the first step in web scraping, allowing access to the webpage's source code.

Extracting Data: Found and extracted movie titles for The 100 Greatest Movies from a webpage using findAll and specified a class to target the appropriate elements. This technique is useful for gathering specific data from a structured webpage.

Manipulating Data: Used Python's list and string methods to process the extracted movie titles, reverse the list, and join the items into a single string for easy output.

Writing Data to a File: Stored the scraped movie titles into a .txt file using Python's file handling functions, allowing for further use or analysis offline.

Project Summary:

In this project, the focus was on scraping the list of the 100 greatest movies from a webpage using BeautifulSoup. The project involved sending an HTTP request to retrieve the HTML content, parsing it with BeautifulSoup, extracting the desired data (movie titles), and saving the results to a text file. This exercise reinforced the basics of web scraping, data extraction, and working with HTML content programmatically.

Day 46: Web Scraping and Spotify Playlist Creation

Key Learnings:

Combining Web Scraping and API Integration: Learned how to combine web scraping with external APIs. The project involved scraping Billboard's Hot 100 list and using the Spotify API to create a playlist based on the scraped data.

Web Scraping with BeautifulSoup: Used BeautifulSoup to extract song titles from a specific Billboard Hot 100 chart page. This required parsing the HTML content and selecting the relevant elements to gather song titles.

Interacting with Spotify API: Used the spotipy library to interact with the Spotify API. Learned how to authenticate with Spotify, search for songs by title, and add them to a new playlist.

Error Handling: Implemented error handling to manage missing songs or API issues, ensuring the program continues to function even when a song is not found on Spotify.

Creating and Modifying Spotify Playlists: Gained hands-on experience in creating a private playlist on Spotify and adding tracks to it programmatically based on the search results from the Billboard list.

Project Summary:

In this project, the goal was to create a Spotify playlist based on the top 100 songs from a specific date on the Billboard Hot 100 chart. The project combined web scraping (to extract song titles) and API integration (to search for those songs on Spotify and create a playlist). The final output was a private playlist on Spotify containing the top tracks from the selected date, showcasing how to use BeautifulSoup for data extraction and Spotipy for interacting with Spotify's music library.

Day 47: Price Tracker with Email Alerts

Key Learnings:

Web Scraping for Price Monitoring: Used BeautifulSoup to scrape a product's price and title from a webpage, learning how to identify and extract specific elements using HTML attributes like id and class.

Price Comparison Logic: Implemented logic to compare the scraped price with a target price to determine whether to trigger an alert.

Automated Email Alerts:
Utilized the smtplib library to send email notifications when the price drops below the target.
Constructed email messages using MIMEText and MIMEMultipart for customizable subject lines and message bodies.

Environment Variables for Security: Leveraged the dotenv library to securely store and access sensitive credentials, such as email address, password, and SMTP server details.

Error Handling in Email Sending: Added error handling to manage exceptions during the email sending process, ensuring the program provides meaningful feedback if something goes wrong.

Project Summary:

The goal of this project was to create a price tracking tool that monitors the price of a product on a website and sends an email alert if the price falls below a predefined threshold. By combining web scraping and email automation, this project demonstrated how to build a practical application for tracking discounts and deals online.

Day 48: Automated Cookie Clicker Bot with Selenium

Key Learnings:

Web Automation with Selenium:
Used the Selenium library to interact with the Cookie Clicker game, automating clicks on the main cookie and other elements like products and upgrades. Practiced using By for locating elements via ID, CLASS_NAME, and XPATH.

Explicit Waits:
Leveraged WebDriverWait with expected_conditions to handle elements that load dynamically, ensuring the bot interacts with elements only after they are visible or clickable.

Real-time Decision Making:
Implemented logic to periodically evaluate available upgrades and products, prioritizing clicks based on price and availability.

Handling Popups/Banners:
Added error handling to detect and close any unwanted banners or popups that may interrupt the automation process.

Performance Measurement:
Calculated cookies per second (cps) as a metric to evaluate the bot's effectiveness after a defined runtime.

Project Summary:

This project automated gameplay for the popular Cookie Clicker game, focusing on maximizing efficiency by:

- Clicking the main cookie continuously.
- Purchasing upgrades and products when available.
- Handling unexpected banners or interruptions gracefully.
- By simulating gameplay for 5 minutes, the bot demonstrated how Selenium could be used for automated testing or gaming experiments. The final cookies-per-second (cps) metric served as a measure of the bot's performance.

Day 49: LinkedIn Job Application Automation with Selenium

The goal of this project was to automate the job application process on LinkedIn, specifically using the "Easy Apply" feature for job listings that match specific criteria.

Process:

Logging in: Automating the login process using stored credentials loaded via environment variables, ensuring secure login without hardcoding sensitive information.
Navigating to Job Listings: Programmatically accessing a job search URL that filters results for data engineering roles.
Automating the "Easy Apply" Process: The automation clicked through multiple steps, such as applying to a job, filling out information, reviewing the application, and submitting it. Additionally, the process also included dismissing unnecessary pop-ups (like "No thanks" prompts).

Error Handling: I implemented error handling to ensure smooth execution, even in cases where elements were not immediately visible or clickable.
Key Learnings:

Conclusion:
This project demonstrated the power of automation in simplifying tedious tasks like applying to job listings. By combining web scraping and browser automation, I was able to streamline the process, allowing for more efficient job applications. This experience deepened my knowledge of Selenium and Python for automating web interactions and handling dynamic content.

Day 50: Data entry Automation :

The goal of this project was to scrape real estate property listings from a website and automate the process of filling out a Google Form with the scraped data. This involved using web scraping to extract property details and automation to populate a form with this information.

Process:

Web Scraping:
using BeautifulSoup and requests to scrape data from a real estate website. The data extracted from the listing page and stored in a dictionary with the property index as the key and a list containing the address, price, and link as the value.

Form Automation:
Using Selenium WebDriver, automated the process of filling out a Google Form with the scraped data. The script:

- Opened a pre-configured Google Form.
- Filled out the form fields with the scraped property details.
- Submitted the form after populating it with the relevant information for each property. This was repeated for each property in the scraped data.

Conclusion:

This project was a great exercise in integrating web scraping with browser automation. It provided insights into how these techniques can work together to automate real-world tasks, like filling out forms with data gathered from websites. By automating both data extraction and entry, I was able to improve efficiency and accuracy in repetitive tasks. This project further enhanced my skills with Python, Selenium, and web scraping.