🐍 Python Notes: Days 41-50 (Angela Yu's 100 Days of Code)

This section marks a shift from pure Python programming to web technologies. You'll learn the language of the web (HTML/CSS) to understand how websites are built, then use Python libraries like **Beautiful Soup** and **Selenium** to scrape data and automate browser actions.

Day 41: Introduction to Web Foundations with HTML

🎯 **Goal:** Understand the basic structure and syntax of HTML (HyperText Markup Language). This is not Python, but knowing HTML is essential for web scraping.

Key Concepts

- **What is HTML?** It's the standard markup language used to create web pages. It describes the structure of a web page using elements.
- **HTML Boilerplate:** The essential structure every HTML file needs.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Website</title>
</head>
<body>
    </body>
</html>
```

- **HTML Tags:** Keywords surrounded by angle brackets that define how your web browser will format and display the content.
  - `<h1>` to `<h6>` : Headings.
  - `<p>` : Paragraph.
  - `<em>` or `<i>` : Emphasized or italicized text.
  - `<strong>` or `<b>` : Important or bold text.
  - `<br>` : Line break.
  - `<hr>` : Horizontal rule (a thematic break).

- **Lists:**
  - `<ul>` : Unordered List (bullet points).
  - `<ol>` : Ordered List (numbered).
  - `<li>` : List Item (used inside `<ul>` or `<ol>`).
- **Nesting:** Placing tags inside other tags. Proper nesting is crucial for a valid HTML structure.

```
<p>This is a <strong>bold</strong> paragraph.</p>
```

Project Focus: Personal Site

You'll start building a simple personal website using only HTML. This forces you to get comfortable with structuring content using different tags before you add any styling.

Day 42: Intermediate HTML

🎯 **Goal:** Learn more advanced HTML elements for creating complex layouts and embedding content.

Key Concepts

- **Image Tag:** `<img>` is a self-closing tag used to embed images.
  - `src` : The source attribute, which is the path to the image file.
  - `alt` : The alternate text attribute, which describes the image for screen readers and if the image fails to load.

```
<img src="images/cat.jpg" alt="A photo of my cat">
```

- **Anchor Tag:** `<a>` creates a hyperlink to other web pages, files, or locations within the same page.
  - `href` : The hyperlink reference attribute, which specifies the URL.

```
<a href="https://www.google.com">Go to Google</a>
```

- **Tables:** Used to display data in rows and columns.
  - `<table>` : The main table container.
  - `<tr>` : Table Row.
  - `<th>` : Table Header (a cell with bold, centered text).
  - `<td>` : Table Data (a standard cell).

```
<table>
  <tr>
    <th>Name</th>
    <th>Role</th>
  </tr>
  <tr>
    <td>Angela</td>
    <td>Instructor</td>
  </tr>
</table>
```

- **Forms:** Collect user input.
  - `<form>` : The container for different types of input elements.
  - `<input>` : Can be text fields, checkboxes, password fields, radio buttons, and submit buttons.
  - `<label>` : A caption for an item in a user interface.

Project Focus: Structuring a CV

You'll use tables and other elements to structure your personal CV/resume in HTML, giving it a clear, organized layout.

Day 43: Introduction to CSS

🎯 **Goal:** Learn how to style HTML elements using CSS (Cascading Style Sheets).

Key Concepts

- **What is CSS?** It's the language we use to style an HTML document. CSS describes how HTML elements should be displayed.
- **Ways to Add CSS:**
  i. **Inline CSS:** Using the `style` attribute directly inside an HTML tag (generally bad practice).

```
<h1 style="color:blue;">Blue Heading</h1>
```

  ii. **Internal CSS:** Placing `<style>` tags within the `<head>` section of the HTML file.

```
<head>
  <style>
    body {
      background-color: lightblue;
    }
  </style>
</head>
```

iii. **External CSS:** Linking to a separate `.css` file (best practice for organization).

```
<head>
  <link rel="stylesheet" href="styles.css">
</head>
```

- **CSS Selectors:** Patterns used to select the element(s) you want to style.
  - **Tag Selector:** Selects all elements with that tag name (e.g., `p`, `h1`, `body`).
  - **Class Selector:** Selects elements with a specific class attribute. Starts with a dot (`.`).

```
.highlight { color: yellow; }
```

  - **ID Selector:** Selects one unique element with a specific id attribute. Starts with a hash (`#`).

```
#main-title { font-size: 40px; }
```

Project Focus: Styling Your Personal Site

You'll take the HTML site you built and apply CSS to change colors, fonts, and spacing, making it visually appealing.

Day 44: Intermediate CSS

🎯 **Goal:** Dive deeper into CSS for creating professional layouts, focusing on the Box Model and positioning.

Key Concepts

- **The Box Model:** Every HTML element is essentially a box. This model describes how the different parts of the box (content, padding, border, margin) work together.
  - **Content:** The text, image, or other media in the element.
  - **Padding:** The transparent space around the content, inside the border.
  - **Border:** A line that goes around the padding and content.

    - **Margin:** The transparent space around the border, separating it from other elements.
- **Positioning:** Controlling where elements appear on the page.
    - `static` : Default value. Elements render in order, as they appear in the document flow.
    - `relative` : The element is positioned relative to its normal position.
    - `absolute` : The element is positioned relative to its nearest positioned ancestor.
    - `fixed` : The element is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled.
    - `sticky` : A hybrid of `relative` and `fixed` .
- **Floats and Clears:** An older method for layout, often used to wrap text around images. (Modern layouts often use Flexbox or Grid instead).

Project Focus: Advanced Personal Site Layout

You'll refactor your personal site's CSS to use more advanced positioning and box model properties, creating a more complex and responsive layout.

Day 45: Web Scraping with Beautiful Soup

🎯 **Goal:** Use Python to parse HTML from a live website and extract useful information. This is where your Python and new web knowledge combine! 🚀

Key Concepts

- **Web Scraping:** The process of automatically extracting data from websites.
- **Libraries:**
    - `requests` : A simple and elegant Python library for making HTTP requests to get the HTML content from a URL.
    - `BeautifulSoup` : A library for pulling data out of HTML and XML files. It creates a parse tree that makes it easy to navigate, search, and modify the tree.
- **Workflow:**
    i. **Install libraries:** `pip install requests beautifulsoup4`
    ii. **Fetch the page:** Use `requests.get(URL)` to download the HTML.
    iii. **Create a "soup" object:** `soup = BeautifulSoup(html_content, "html.parser")`
    iv. **Find elements:** Use `soup.find()` , `soup.find_all()` , or `soup.select()` to locate the data you need using tags, classes, or IDs.
    v. **Extract data:** Get the text or attributes from the found elements using `.getText()` or `element['attribute_name']` .

Code Example

```
import requests
from bs4 import BeautifulSoup

URL = "https://news.ycombinator.com/"

response = requests.get(URL)
website_html = response.text

soup = BeautifulSoup(website_html, "html.parser")

# Find the first article title
first_article_tag = soup.find(name="span", class_="titleline").find("a")
article_text = first_article_tag.getText()
article_link = first_article_tag.get("href")

print(f"Title: {article_text}")
print(f"Link: {article_link}")
```

Project Focus: 100 Movies to Watch

You'll scrape a website (like Empire's 100 Greatest Movies) to get a list of movie titles, then save them to a `.txt` file in order.

Day 46: Automating Spotify with Selenium

🎯 **Goal:** Go beyond static scraping. Learn to control a web browser programmatically to perform actions on a dynamic website.

Key Concepts

- **Static vs. Dynamic Websites:**
  - **Static:** The content is fixed and delivered as-is (perfect for Beautiful Soup).
  - **Dynamic:** Content is loaded or changed with JavaScript after the initial page load (e.g., infinite scroll, pop-ups). Beautiful Soup can't handle this because it doesn't run JavaScript.
- **Selenium WebDriver:** A browser automation framework. It allows your Python script to open a browser, navigate to URLs, and interact with page elements just like a human would (clicking buttons, filling forms, etc.).

- **Setup:**
  i. Install Selenium: `pip install selenium`
  ii. Download a `WebDriver` for your browser (e.g., `chromedriver` for Chrome). Make sure its version matches your browser's version.
- **Basic Commands:**
  - `driver = webdriver.Chrome()` : Opens a new Chrome browser window.
  - `driver.get(URL)` : Navigates to a URL.
  - `driver.find_element(By.NAME, "q")` : Finds a single element. You can find by `ID` , `NAME` , `CLASS_NAME` , `CSS_SELECTOR` , etc.
  - `driver.quit()` : Closes the browser window.

Code Example

```
from selenium import webdriver
from selenium.webdriver.common.by import By

# Path to your chromedriver executable
chrome_driver_path = "/path/to/your/chromedriver"
driver = webdriver.Chrome(executable_path=chrome_driver_path)

driver.get("https://www.python.org")

# Find an element by its CSS Selector
search_bar = driver.find_element(By.NAME, "q")
print(search_bar.get_attribute("placeholder")) # Prints "Search"

driver.quit()
```

Project Focus: Spotify Playlist Bot

You will use the `spotipy` library to authenticate with Spotify and Selenium to scrape the Billboard Hot 100 for a specific date, then create a new Spotify playlist with all those songs.

Day 47: Amazon Price Tracker Bot

🎯 **Goal:** Combine web scraping and browser automation to build a useful tool that monitors a product price and notifies you.

Key Concepts

- **Finding Elements with Selenium:** Mastering different `By` strategies is key.
  - `By.ID` : Good for unique elements.
  - `By.NAME` : Often used for form inputs.
  - `By.XPATH` : A powerful way to navigate the HTML structure, even for complex paths.
  - `By.CSS_SELECTOR` : Very flexible and often more readable than XPath.
- **Extracting Text:** Once you have an element, use the `.text` attribute to get the visible text content.
- **Data Cleaning:** The text you scrape is often messy (e.g., `$19.99\n(Sale)`). You'll need to use Python string methods like `.strip()` , `.split()` , and type casting (`float()`) to clean it up into a usable number.
- **Sending Emails with Python:**
  - Use the built-in `smtplib` library to connect to an email server (like Gmail, Outlook) and send an email. This is how your bot will notify you.

Project Focus: Amazon Price Tracker

Your script will:

1. Use Selenium to navigate to an Amazon product page.
2. Scrape the current price of the item.
3. Clean the price data and convert it to a float.
4. Compare the current price to your target price.
5. If the price is at or below your target, use `smtplib` to send you an email alert.

Day 48: Advanced Selenium & Game Playing Bot

🎯 **Goal:** Learn how to handle websites that take time to load content and how to perform more complex user actions.

Key Concepts

- **Waiting Strategies:** A major challenge in automation is that scripts can run faster than a webpage loads. If your script tries to find an element before it exists, it will crash.
  - **Implicit Wait:** `driver.implicitly_wait(5)` tells Selenium to wait up to 5 seconds for an element to appear before throwing an error. It's a global setting.

- **Explicit Wait (Preferred):** More precise. You tell Selenium to wait for a *specific condition* to be met. This is more reliable.

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# Wait up to 10 seconds until the element with ID 'myElement' is clickable
element = WebDriverWait(driver, 10).until(
    EC.element_to_be_clickable((By.ID, "myElement"))
)
```

- **ActionChains:** For performing complex actions like hovering, drag-and-drop, or key presses.
- **Sending Keystrokes:** The `.send_keys()` method can be used to type into input fields or to send special keys like `Keys.ENTER`.

Project Focus: Cookie Clicker Bot

You'll create a bot that plays the Cookie Clicker game. It will:

1. Repeatedly click the giant cookie as fast as possible.
2. After a set interval (e.g., every 5 seconds), it will check the store for affordable upgrades.
3. It will buy the most expensive affordable upgrade to maximize cookie production.
4. This project heavily relies on managing time and checking for interactable elements.

Day 49: Automating Job Applications on LinkedIn

🎯 **Goal:** Apply your Selenium skills to a real-world, multi-step workflow: automating job applications.

Key Concepts

- **Workflow Automation:** Breaking down a complex task into a series of discrete, repeatable steps that your script can execute.
  - Step 1: Go to LinkedIn jobs page.
  - Step 2: Sign in.
  - Step 3: Search for jobs with specific keywords and filters.
  - Step 4: Iterate through the search results.
  - Step 5: For each job, click the "Easy Apply" button.
  - Step 6: Fill out the application form (phone number, resume).
  - Step 7: Submit the application.

- **Handling Pop-ups and New Windows:** Sometimes clicking a link opens a new tab or a modal window. You need to learn how to switch the `driver` 's focus to the new context.
- **Error Handling:** What happens if a job isn't "Easy Apply"? Or if a field is missing? Using `try...except` blocks is crucial for making your bot robust so it doesn't crash on the first unexpected event.

Project Focus: LinkedIn Job Application Bot

You will build a bot that can log in to LinkedIn, search for jobs using filters you define, and automatically complete the "Easy Apply" process for each listing it finds. **Disclaimer:** Use this ethically and sparingly. The goal is the learning exercise, not spamming recruiters.

Day 50: Auto Tinder Swiping Bot

🎯 **Goal:** Build another complex automation bot that deals with pop-ups, dynamic content, and making decisions based on scraped information.

Key Concepts

- **Consolidating Skills:** This project combines everything you've learned about Selenium:
  - Logging into a site with credentials.
  - Using explicit waits to handle pop-ups (location access, notifications, etc.).
  - Finding elements using complex CSS selectors or XPath.
  - Sending clicks to perform actions (swiping left or right).
  - Creating a loop to perform the main action repeatedly.
  - Using `try...except` to gracefully handle interruptions like running out of likes or getting a match.
- **Decision Logic:** The bot needs a simple rule to follow, e.g., "swipe right on everyone" or a more complex (and likely ineffective) rule based on limited profile info. The focus is on the automation, not the decision-making.

Project Focus: Tinder Swiping Bot

The goal is to create a bot that logs into Tinder, dismisses all the initial pop-ups, and then enters a loop to automatically swipe right on profiles. It's a fun and challenging exercise in controlling a modern, dynamic web application.