Week 2: HTML and Git/GitHub

In **Week 2**, we continue building foundational skills in HTML, focusing on structuring content with lists, tables, and forms. These elements are critical for displaying organized data and gathering input from users.

1. **Creating Lists: Organizing Content Using `<ol>`, `<ul>`, and `<li>`**

HTML provides two main types of lists that help in presenting grouped information: ordered and unordered lists. Both are made up of list items (`<li>`), but they serve different purposes based on whether the order of the items is significant or not.

a. **Ordered Lists (`<ol>`)**

An ordered list displays items in a specific sequence, typically numbered. This is useful when the order of items matters, such as in instructions or steps.

```
<ol>
  <li>Step 1: Preheat the oven to 350°F.</li>
  <li>Step 2: Mix the ingredients.</li>
  <li>Step 3: Bake for 30 minutes.</li>
</ol>
```

In the example above, the list items (`<li>`) are numbered automatically, with the browser generating a number for each list item.

b. **Unordered Lists (`<ul>`)**

An unordered list displays items as a bulleted list, where the sequence doesn't matter. This is commonly used for grouping related items or options.

```
<ul>
  <li>Apples</li>
  <li>Bananas</li>
  <li>Oranges</li>
</ul>
```

You can also nest lists to create more complex structures:

```
<ul>
  <li>
    Fruits
    <ul>
      <li>Apples</li>
```

```
      <li>Bananas</li>
    </ul>
  </li>
  <li>Vegetables</li>
</ul>
```

c. **Key Takeaways**

- **Ordered Lists**: Items have a meaningful order (e.g., instructions, rankings).
- **Unordered Lists**: Items are grouped without needing a specific order (e.g., bullet points).
- **List Items (`<li>`)**: Define each individual item in both ordered and unordered lists.

2. **Tables: Structuring Data with `<table>`, `<tr>`, `<th>`, and `<td>`**

Tables allow you to organize data into rows and columns, making it easy to present structured information. HTML tables are often used for displaying schedules, comparisons, or any data in tabular format.

a. **Basic Table Structure**

A table is created using the `<table>` tag. Inside, rows are defined with `<tr>`, and within each row, cells are created with either `<th>` for header cells (bold, centered by default) or `<td>` for regular data cells.

```
<table>
  <tr>
    <th>First Name</th>
    <th>Last Name</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>30</td>
  </tr>
  <tr>
    <td>Jane</td>
    <td>Smith</td>
    <td>25</td>
  </tr>
</table>
```

This example creates a simple table with two rows of data and a header row. The `<th>` elements define the headers (First Name, Last Name, Age), and the `<td>` elements contain the actual data.

b. **Table Headers (`<thead>`) and Body (`<tbody>`)**

For more complex tables, you can separate the header section from the body for better structure and clarity:

```
<table>
  <thead>
    <tr>
      <th>Product</th>
      <th>Price</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Laptop</td>
      <td>$999</td>
    </tr>
    <tr>
      <td>Phone</td>
      <td>$599</td>
    </tr>
  </tbody>
</table>
```

c. **Styling Tables**

You can enhance tables using CSS to add borders, adjust spacing, or even create alternating row colors for readability.

d. **Key Takeaways**

- **Tables**: Ideal for displaying structured data.
- **Rows (`<tr>`)**: Define horizontal rows of content.
- **Header Cells (`<th>`)**: For column headers (bold and centered by default).
- **Data Cells (`<td>`)**: For regular table data.
- `<thead>` **and** `<tbody>`: Separate the table header from the body for clarity.

3. **Forms and Input Fields: Gathering User Data**

Forms are essential for collecting information from users. HTML provides various form elements that enable users to input data, submit it, and interact with websites. This section covers the basics of forms and their input types.

a. **Form Basics: The `<form>` Tag**

The `<form>` element is the container for all input elements. It defines how data will be sent (action) and which method will be used (GET or POST).

```
<form action="/submit" method="POST">
  <!-- Form inputs go here -->
</form>
```

b. **Common Form Elements**

- **Text Input (`<input type="text">`)**: A basic text field where users can enter a single line of text.

```
<label for="name">Name:</label> <input type="text" id="name" name="name"
/>
```

- **Password Input (`<input type="password">`)**: Similar to text input, but the characters are masked.

```
<label for="password">Password:</label>
<input type="password" id="password" name="password" />
```

- **Email Input (`<input type="email">`)**: A specialized field that checks for valid email formats.

```
<label for="email">Email:</label>
<input type="email" id="email" name="email" />
```

- **Number Input (`<input type="number">`)**: Restricts the input to numerical values.

```
<label for="age">Age:</label> <input type="number" id="age" name="age" />
```

c. **Textarea (`<textarea>`)**

For multi-line text input, the `<textarea>` tag is used. Unlike the `<input>` element, the `<textarea>` allows users to enter larger blocks of text.

```
<label for="message">Message:</label>
<textarea id="message" name="message" rows="4" cols="50"></textarea>
```

d. **Select Dropdown (`<select>`)**

The `<select>` element allows users to choose from a list of predefined options.

```
<label for="country">Country:</label>
<select id="country" name="country">
  <option value="us">United States</option>
  <option value="ca">Canada</option>
</select>
```

e. **Checkboxes and Radio Buttons**

- **Checkbox (`<input type="checkbox">`)**: Allows users to select multiple options from a list.

```
<label
  ><input type="checkbox" name="subscribe" value="newsletter" />
Subscribe to
  newsletter</label
>
```

- **Radio Buttons (`<input type="radio">`)**: Used when only one option can be selected from a group.

```
<label><input type="radio" name="gender" value="male" /> Male</label>
<label><input type="radio" name="gender" value="female" /> Female</label>
```

f. **Key Takeaways**

- **Forms**: Containers for user input that send data to a server.
- **Input Types**: Various fields for different kinds of data—text, password, email, number, etc.
- **Select and Textarea**: Useful for dropdowns and multi-line text input.
- **Checkboxes and Radio Buttons**: For single or multiple option selection.

Week 2: HTML and Git/GitHub (Continued)

In **Week 2**, after gaining proficiency in HTML fundamentals, students will now explore how to publish their HTML projects using **GitHub Pages**, and deepen their understanding of **advanced Git concepts** such as branching, merging, and conflict resolution.

1. Using GitHub Pages for Hosting Simple HTML Websites

GitHub Pages is a free hosting service provided by GitHub, allowing developers to quickly deploy static websites. By the end of this section, students will know how to push their projects to GitHub and configure GitHub Pages to make their website publicly accessible via a URL.

a. What is GitHub Pages?

GitHub Pages is a feature of GitHub that allows users to host their web projects directly from a repository. It's ideal for static websites (HTML, CSS, JavaScript) and personal projects. This feature enables developers to showcase their work without needing a separate hosting service.

b. Steps to Host a Simple HTML Website Using GitHub Pages

1. **Create a Repository on GitHub**:
    - Log in to GitHub and create a new repository for your project.

- Give the repository a meaningful name (e.g., `my-first-website`).

2. **Push Code to GitHub**:
   - Once the repository is created, clone it to your local machine or initialize it with `git init`.
   - Add your HTML files (e.g., `index.html`, `style.css`, etc.) to the local repository.
   - Commit the changes and push them to GitHub:

```
git add .
git commit -m "Initial commit"
git push origin main
```

3. **Configure GitHub Pages**:
   - Go to the "Settings" tab of your GitHub repository.
   - Scroll down to the **GitHub Pages** section.
   - Select the branch you want to use (usually `main` or `master`), and save the settings.

4. **Access the Website**:
   - After a few moments, GitHub will deploy your site. You can access it through the URL: `https://<username>.github.io/<repository-name>`.

c. GitHub Pages Key Takeaways

- GitHub Pages allows for free and simple hosting of static websites.
- The process involves creating a repository, pushing code to GitHub, and configuring GitHub Pages to make the site public.
- This feature is ideal for showcasing personal portfolios, documentation, or small projects.

2. Advanced Git Concepts: Branching, Merging, and Resolving Conflicts

In this section, students will explore some of Git's more powerful features, such as **branching** and **merging**. These concepts allow developers to work on separate parts of a project independently and later combine their work without disrupting the main codebase.

a. Branching in Git

Branching is a fundamental concept in Git that allows developers to diverge from the main codebase (typically called `main` or `master`) to work on different features or fixes. This is especially useful when multiple developers or teams are working on a project.

1. **Creating a Branch**:
   - To start a new branch, use the `git branch` command followed by the branch name:

```
git branch new-feature
```

2. **Switching Between Branches**:
    - To switch to the new branch and start working on it, use:

```
git checkout new-feature
```

3. **Making Changes on the Branch**:
    - After switching to the new branch, students will make their changes, such as adding new features or bug fixes, and then commit those changes.

b. Merging Branches

Once a feature has been completed in a separate branch, students will learn how to merge that branch back into the main branch (`main` or `master`). This allows them to integrate their changes with the main codebase.

1. **Merging with the Main Branch**:
    - First, switch to the main branch:

```
git checkout main
```

    - Then, merge the feature branch into the main branch:

```
git merge new-feature
```

    - Git will automatically merge changes from both branches unless there are conflicting changes.

c. Resolving Merge Conflicts

Sometimes, when merging branches, Git encounters conflicts, meaning changes from different branches are incompatible. These conflicts must be manually resolved by the developer.

1. **What is a Merge Conflict?**:
   A merge conflict occurs when two branches modify the same part of a file in different ways, and Git cannot automatically reconcile the differences.

2. **How to Resolve Conflicts**:

- **VS Code** provides built-in Git tools to help resolve conflicts.
- Conflicted files will be marked, and inside the file, Git will show the conflicting changes using `<<<<<<<`, `=======`, and `>>>>>>>` markers.

Example of a conflict:

```
<<<<<<< HEAD
<h1>Welcome to My Website</h1>
=======
<h1>Welcome to My Portfolio</h1>
>>>>>>> new-feature
```

- The developer must decide which version to keep or manually merge the changes.
- After resolving the conflicts, remove the conflict markers and save the file.

3. **Finalizing the Merge**:
   Once conflicts are resolved, the developer commits the changes:

```
git add conflicted-file.html
git commit -m "Resolved merge conflict"
```

d. Merging Branches Key Takeaways

- **Branching**: Allows developers to work on different features independently, preventing changes from interfering with each other.
- **Merging**: Combines branches into the main codebase, integrating changes from different developers or features.
- **Merge Conflicts**: Occur when changes made on different branches clash. Students will learn how to resolve conflicts manually in VS Code.

In **the final part of Week 2**, students will dive into **collaborative coding with GitHub**, learning essential GitHub features that facilitate teamwork and open-source contribution. These skills are crucial for real-world development projects, where multiple developers work together on a single project.

1. Collaborative Coding with GitHub

In this section, students will learn how to collaborate with others using GitHub by cloning repositories, forking repositories, and submitting pull requests. These features allow for seamless teamwork, even in large projects with many contributors.

a. Cloning Repositories

Cloning is a method used to copy a repository from GitHub to your local machine. This allows developers to work on a project without needing to have direct write access to the original repository.

1. **Why Clone a Repository?**
    - Developers clone repositories to work on a project locally without altering the original codebase.
    - This is useful for contributing to open-source projects or collaborating on private team projects.
2. **How to Clone a Repository**:
    - In GitHub, navigate to the repository you want to clone.
    - Click the green "Code" button, and copy the URL of the repository (either HTTPS or SSH).
    - In your terminal, use the `git clone` command followed by the repository URL:

```
git clone https://github.com/username/repository-name.git
```

3. **Key Takeaways**:
    - Cloning a repository allows you to create a local copy of the project, where you can make changes and push updates.
    - This is a fundamental skill when working with both open-source projects and team collaborations.

b. Forking and Pull Requests

Forking is a way to create a personal copy of someone else's GitHub repository. After making changes to the forked version, students can submit a **pull request** (PR) to propose their changes to the original repository. This process is commonly used in open-source collaboration.

1. **What is Forking?**
    - Forking a repository creates a copy of the project under your GitHub account.
    - This is useful for contributing to open-source projects where you don't have direct write access to the original repository.
2. **How to Fork a Repository**:
    - On GitHub, find the repository you want to contribute to.
    - Click the "Fork" button in the top-right corner of the repository page.
    - The repository will be copied to your GitHub account, and you can clone it to your local machine to start working.
3. **Creating a Pull Request (PR)**:
    - After forking and making changes to your local copy, push the changes to your forked repository on GitHub.
    - Then, navigate to the original repository and click on the "Pull Requests" tab.

- Click the "New Pull Request" button and select your forked repository as the source, and the original repository as the target.
- GitHub will show the differences between the two versions and allow the repository owner to review and merge your changes.

4. **Best Practices for Pull Requests**:
    - Keep pull requests focused on a single task or feature to make it easier for others to review.
    - Provide a clear description of what the pull request addresses and why changes are necessary.
    - Ensure your code follows the project's coding style and includes necessary tests, if applicable.
5. **Key Takeaways**:
    - Forking is a method for contributing to open-source projects when you don't have direct access to the original codebase.
    - Pull requests allow developers to propose changes to a project, enabling collaboration and review by others before merging into the main codebase.

Final Thoughts for Week 2:

By the end of **Week 2**, students will have gained a solid foundation in both **HTML** and **Git/GitHub**. This week's lessons provided essential skills for building static websites with HTML and understanding the fundamental concepts of **version control** and **collaborative coding**.

Key Takeaways:

- **HTML Basics**:
    - Students learned the fundamental structure of an HTML document, including how to create basic webpage structures using elements such as `<html>`, `<head>`, `<body>`, and more. They also gained knowledge of **semantic HTML** elements that improve accessibility and SEO, such as `<header>`, `<main>`, and `<footer>`.
    - They practiced creating common HTML elements like **headings**, **paragraphs**, **links**, and **images**, understanding how to use attributes (like `src` and `alt`) for images and `href` for links.
- **Lists, Tables, Forms, and Input Types**:
    - Students explored **lists** (`<ul>`, `<ol>`) and learned how to create organized, easy-to-read content.
    - They learned how to structure **tables** using `<table>`, `<tr>`, `<th>`, and `<td>`, and built simple tables to organize data.
    - The session on **forms** introduced students to gathering user data through **input fields** like `<input>`, `<select>`, and `<textarea>`. They practiced working with different input types such as text, email, password, checkboxes, and radio buttons.
- **Using GitHub Pages for Hosting Simple HTML Websites**:

- Students learned how to **publish their HTML projects** using **GitHub Pages**, which provided them with a free hosting solution to showcase their work online.
- They practiced pushing their code to GitHub, configuring the repository for GitHub Pages, and making a simple HTML site accessible via a public URL.

- **Advanced Git Concepts**:

  - **Branching and Merging**: Students practiced creating **branches** to work on new features without affecting the main codebase. They also learned how to **merge** branches back into the main branch after completing the feature or fix, practicing the workflow that is used in collaborative development environments.
  - **Resolving Merge Conflicts**: Students were introduced to **merge conflicts**—what they are and how to resolve them. They practiced resolving conflicts using **VS Code**'s built-in Git tools, which will be essential when working with team members and managing code that has diverged.

- **Collaborative Coding with GitHub**:

  - **Cloning Repositories**: Students learned how to **clone** repositories from GitHub to contribute to projects, enabling them to collaborate with others and work on open-source projects.
  - **Forking and Pull Requests**: The concept of **forking** was introduced, showing students how to make a copy of a repository and propose changes to the original project via **pull requests**.
  - This skill set equips students for real-world development practices, where working on collaborative projects and contributing to open-source codebases are common tasks.

Outcomes by Week's End

By the end of **Week 2**, students will:

1. Be comfortable building basic HTML websites using proper structure and semantic tags.
2. Understand how to organize and display data using lists, tables, and forms.
3. Be capable of hosting their HTML projects using **GitHub Pages**.
4. Have gained confidence in **Git** basics, including committing changes and managing project versions.
5. Understand how to work with branches, merge them, and resolve conflicts.
6. Be able to collaborate on coding projects using **GitHub**, through cloning, forking, and submitting pull requests.

These skills provide students with the foundational knowledge required for both **front-end web development** and collaborative software development. They're now ready to start contributing to open-source projects, working with GitHub in team environments, and building dynamic web pages.