Month 1: Web Foundations

Week 1: Introduction to Web Development

1. **Overview of the Web: How Websites Work**

To build websites effectively, it's crucial to understand how the web operates. The web consists of a vast network of computers and devices communicating with each other to serve websites. Here's how it works:

- **Client-Server Model**:
  At the heart of web communication is the client-server model. The **client** is typically your web browser (e.g., Chrome, Firefox, Safari), which sends a request to a **server** to retrieve a webpage. The server stores the website's resources (HTML, CSS, JavaScript, images, etc.) and delivers them back to the client.

  > Example: When you type `www.google.com` into your browser, the browser sends a request to Google's server. The server processes that request and responds by sending back the HTML, CSS, and other resources that make up the Google homepage.

- **Resources that Make a Webpage**:
  - **HTML (Hypertext Markup Language)**: This is the structure of the webpage, like the bones of a body. It defines the elements that make up the page (e.g., headings, paragraphs, images, links).
  - **CSS (Cascading Style Sheets)**: CSS styles the page, adding design and layout elements (e.g., colors, fonts, spacing).
  - **JavaScript**: Adds interactivity to a webpage, enabling dynamic content like slideshows, form validation, and data fetching.

2. **Understanding Web Architecture**

In this section, students will dive deeper into how websites are structured and the flow of data between the client and server.

- **Client-Server Communication**:
  When you request a website (e.g., typing in a URL), your browser acts as the **client**, and the computer hosting the website is the **server**. Here's a simplified version of the process:

  1. **Request**: The browser sends an HTTP request to the server, asking for the webpage.
  2. **Response**: The server responds with the HTML, CSS, and JavaScript files.
  3. **Rendering**: The browser interprets the files and renders the webpage for the user to interact with.

  > **Key Terms**:
  - **HTTP/HTTPS**: The protocol used for communication between the client and server. HTTPS is the secure version.

    - **DOM (Document Object Model)**: After the browser receives HTML from the server, it converts it into a structure called the DOM. This is an in-memory tree representation of the webpage that JavaScript can manipulate to change the content and structure dynamically.

- **Essential Web Development Tools**:

    - **HTML, CSS, and JavaScript**: These form the foundation of front-end development.
    - **APIs (Application Programming Interfaces)**: Front-end developers often fetch data from servers using APIs, allowing web pages to display dynamic content.

3. **Setting Up the Development Environment**

Before coding, it's important to set up a well-organized development environment that boosts productivity.

- **Installing and Configuring VS Code**:
  Visual Studio Code (VS Code) is a widely-used code editor that's favored for its simplicity and powerful features. Here's how to get started:

    1. **Installation**: Download VS Code from the official website. It supports Windows, macOS, and Linux.
    2. **Basic Setup**: Once installed, you can personalize VS Code to fit your workflow. Select a theme that suits your style, and configure your settings to auto-save files or show line numbers for better visibility.
    - **Must-Have Extensions**:
      VS Code's extensibility is one of its greatest strengths. These extensions will make your development process smoother:
        - **Live Server**: This extension enables you to launch a local development server that automatically refreshes the webpage in the browser every time you save a file. This provides immediate feedback on your changes.
        - **Prettier**: A code formatter that helps maintain a clean and consistent coding style. It automatically formats your code every time you save a file, reducing errors and improving readability.
        - **ESLint**: A tool that highlights potential errors or issues in your JavaScript code. It enforces best practices, making your code cleaner and less prone to bugs.

> **Practical Activity**:

    - After installing VS Code, practice creating a basic HTML file and use Live Server to see how the page loads in the browser.

4. **Browser Developer Tools**

Every modern browser comes equipped with powerful **Developer Tools** that allow you to inspect, test, and debug web pages directly in the browser. Understanding these tools is essential for diagnosing issues and improving your code.

- **Inspecting the DOM**:
  When viewing a webpage, you can right-click any element and select "Inspect" to see the

underlying HTML structure. This opens the **Elements** panel of Developer Tools, where you can:

  - View the entire DOM (Document Object Model).
  - Inspect and modify individual HTML elements in real-time.
  - Add or change styles (CSS) directly within the browser.

> Example: Right-click on a header in a website, inspect it, and try changing its text or color through Developer Tools.

- **Modifying CSS in Real-Time**:
  The **Styles** panel allows you to change CSS rules on the fly. You can see immediate results without having to change your actual code. This is especially useful when debugging layout or design issues.

- **Analyzing Network Requests**:
  The **Network** panel provides insight into how and when resources are loaded. You can:

  - Track each request the browser makes to the server.
  - See how long it takes for the server to respond.
  - Identify issues such as slow-loading assets or failed requests.

> Practical Tip: Open a website, and use the Network tab to examine the resources being loaded. Identify any large files that might be slowing down the website's performance.

- **JavaScript Debugging**:
  In the **Console** panel, you can see any errors or warnings generated by your JavaScript code. You can also run custom JavaScript commands, helping you test and troubleshoot issues directly from the browser.

> Practice: Use the Console to log messages (`console.log("Hello, world!")`) and check the output.

5. **The Role of a Front-End Developer**

Front-end development focuses on the **visual and interactive parts** of a website. As a front-end developer, your job is to take designs (often provided by UI/UX designers) and implement them using HTML, CSS, and JavaScript. Here's a breakdown of your responsibilities:

- **Building User Interfaces (UIs)**:
  Front-end developers are responsible for creating the elements users interact with on a website. This could be buttons, forms, navigation menus, or any other component that makes up the interface.

- **Creating Responsive Designs**:
  With the growing number of devices and screen sizes, front-end developers must ensure that websites adapt to different viewports. This is done using techniques like **media queries** and **flexbox** in CSS, which allow layouts to adjust based on the screen's dimensions.

> Example: Designing a website that looks good on both a mobile phone and a large desktop monitor.

- **Ensuring Accessibility**:
  Accessibility is about ensuring that everyone, including people with disabilities, can use your website. This might involve adding alternative text for images, ensuring all interactive elements are keyboard-accessible, and following best practices like proper heading structure.

- **Optimizing User Experience (UX)**:
  UX is a major focus of front-end development. Beyond just building interfaces, a front-end developer must think about how users will interact with the website. This includes:

  - Ensuring fast load times.
  - Reducing unnecessary complexity in interactions (e.g., minimizing the number of steps to complete a task).
  - Making sure the design is intuitive and easy to navigate.

- **Collaboration with Back-End Developers**:
  Front-end developers work closely with back-end developers, who handle the server, database, and application logic. You'll often use **APIs** (Application Programming Interfaces) to retrieve and display data on the front-end, making sure users can interact with dynamic content (e.g., submitting forms, fetching user-specific data).

Key Takeaways

- **Web Architecture**: Websites operate on the client-server model, with the browser requesting resources from a server.
- **Development Tools**: Using VS Code and browser Developer Tools efficiently will greatly improve your coding process and debugging capabilities.
- **Front-End Responsibilities**: As a front-end developer, you'll focus on building user interfaces, ensuring responsive design, optimizing for accessibility, and collaborating with back-end systems.

6. Introduction to Version Control with Git and GitHub

As part of the web development foundations, understanding **version control** is essential for managing changes in your code efficiently. This section introduces **Git** and **GitHub**, two indispensable tools in the development workflow.

a. Git Basics: What is Version Control?

Version control helps developers manage changes to code, track the history of a project, and collaborate with others. **Git** is a widely-used version control system that enables developers to:

- Track changes over time.
- Roll back to previous versions of the project if something goes wrong.
- Collaborate with other developers without conflicts.

> **Example**: Imagine you're working on a new feature and accidentally break the website's layout. With Git, you can revert to a previous commit where everything was working perfectly.

Key Concepts

- **Repository (Repo)**: A Git repository contains all the project files and a history of changes. You can create a local Git repo to begin tracking changes in your project.

  > **Command**: To initialize a new Git repository:

  ```
  git init
  ```

- **Commits**: Commits are snapshots of your project at different points in time. A commit records the state of all your files at that moment.

  > **Command**: To create a commit, you first stage your changes:

  ```
  git add .
  ```

  Then, you commit them with a message describing the changes:

  ```
  git commit -m "Add new navigation menu"
  ```

- **Tracking Changes**: Git keeps track of every file you modify, add, or delete. This allows you to manage your code systematically and revert if needed.

Why Version Control?

- **Backup**: By using Git, you can store your code safely in multiple locations.
- **Collaboration**: Git simplifies working on large projects with multiple developers by keeping track of who made what changes and when.

b. GitHub Overview: Managing Code in the Cloud

While Git operates locally on your computer, **GitHub** provides a cloud-based platform to store your code, collaborate with other developers, and manage your projects online.

- **Creating a GitHub Account**:
  Students will begin by creating an account at GitHub. This account allows them to host repositories, collaborate with others, and even showcase their work.

- **Repositories on GitHub**:
  A **repository** on GitHub serves as the online counterpart to your local Git repository. You can create a new repository on GitHub and link it to your local machine.

  > **Command**: To connect your local repository to a GitHub repository:

  ```

  ```

```
git remote add origin https://github.com/your-username/your-repository-name.git
```

- **Pushing Changes to GitHub**:
  Once connected, you can push your local changes to GitHub, making them accessible online.

  > **Command**: After making a commit locally, push the changes:

  ```
  git push origin main
  ```

GitHub Features

- **Collaboration**: Teams can work together on the same project, sharing code changes through GitHub.
- **Backup**: GitHub serves as a backup of your local repository, ensuring your project is safe in the cloud.

c. Branches and Pull Requests: Parallel Development and Collaboration

One of the powerful features of Git is **branching**. Branches allow you to work on new features or bug fixes without affecting the main version of your code.

- **What is a Branch?**
  A branch in Git is a parallel version of your project where you can make changes independently of the main codebase. For instance, you might create a branch called `feature/homepage-redesign` to redesign the homepage, without affecting the live website.

  > **Command**: To create and switch to a new branch:

  ```
  git checkout -b feature/homepage-redesign
  ```

- **Merging Branches**:
  After completing your work on a feature branch, you can merge it back into the main branch.

  > **Command**: First, switch to the main branch, then merge the feature branch:

  ```
  git checkout main
  git merge feature/homepage-redesign
  ```

Pull Requests on GitHub

When collaborating with others, GitHub's **pull request** system is used to propose code changes. Pull requests allow team members to review and discuss changes before they are merged into the main branch.

- **Creating a Pull Request**:
  After pushing changes to a branch, you can open a pull request on GitHub to propose merging those changes into the main branch.

  > Steps:

  1. Push your changes to GitHub.
  2. Open the repository on GitHub and navigate to the "Pull Requests" section.
  3. Create a new pull request, specifying which branch you want to merge.
  - **Code Review**: Other team members can review the code, make comments, and request changes before approving the pull request.
  - **Approval**: Once approved, the pull request can be merged into the main branch, integrating the new feature or fix.

d. Real-World Scenario: Working with Git and GitHub in Teams

Imagine you are working with a team of front-end developers on a website. Each team member works on different parts of the site. Here's how Git and GitHub help manage the project:

1. **Creating Feature Branches**: Each developer creates their own branch for a specific feature (e.g., `header-redesign`, `new-footer`, `sidebar-improvements`).
2. **Commits and Pushes**: Developers work on their feature, making commits locally, then push their changes to GitHub.
3. **Pull Requests and Review**: Once a developer is done, they open a pull request on GitHub. The team reviews the changes, suggests improvements, and eventually merges the feature into the main branch.
4. **Managing Conflicts**: If two developers made conflicting changes, Git helps by highlighting conflicts so they can be resolved before merging.

GitHub Key Takeaways

- **Version Control**: Git allows you to track changes in your code and collaborate effectively with others.
- **GitHub**: A platform for hosting Git repositories in the cloud, making it easy to share and collaborate on projects.
- **Branches**: Developers can work independently on features or fixes without interfering with the main codebase.
- **Pull Requests**: A structured way to review and merge code changes in a team setting.

7. HTML Basics: Structure, Semantic Elements, Headings, Paragraphs, Links, and Images

HTML (**Hypertext Markup Language**) is the foundation of web pages. It structures content and tells the browser how to display text, images, and links on a web page. This section introduces the basic structure and key elements that make up an HTML document.

a. HTML Structure: Building the Skeleton of a Web Page

Every HTML document follows a specific structure that allows web browsers to interpret and display content. The fundamental elements that make up this structure include:

- **DOCTYPE Declaration**:
  The first line in any HTML file is the `<!DOCTYPE html>` declaration. It tells the browser that the document is written in HTML5, the current standard of the language.

```
<!DOCTYPE html>
```

- **HTML Element (`<html>`)**:
  The entire content of an HTML document is enclosed within the `<html>` tag. This represents the root element of the web page.

```
<html lang="en">
  <!-- Content goes here -->
</html>
```

- **Head Section (`<head>`)**:
  The `<head>` tag contains metadata about the webpage, such as the title (displayed on the browser tab), links to CSS files, and other information that the browser needs to render the page correctly.

```
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
  <title>My First Web Page</title>
</head>
```

- **Body Section (`<body>`)**:
  The `<body>` tag contains the visible content of the web page, including headings, paragraphs, images, and links. All user-visible content is placed within this tag.

```
<body>
  <h1>Welcome to My Website</h1>
  <p>This is a paragraph of text.</p>
</body>
```

Together, these elements form the **skeleton** of every HTML page.

b. Understanding Semantic HTML: Accessible and SEO-Friendly Code

Semantic HTML refers to using elements that describe the meaning of the content they contain. Unlike generic `<div>` and `<span>` elements, **semantic elements** provide more information to both the browser and search engines, which improves **accessibility** and **SEO (Search Engine Optimization)**.

Key Semantic Elements

- `<header>`: Represents the header section of a page or section, typically containing navigation links or introductory content.

```
<header>
  <h1>Site Title</h1>
  <nav>
    <a href="#home">Home</a>
    <a href="#about">About</a>
  </nav>
</header>
```

- `<main>`: Indicates the main content of a webpage, which is unique to that page and central to its purpose.

```
<main>
  <article>
    <h2>Blog Post Title</h2>
    <p>This is the content of the article.</p>
  </article>
</main>
```

- `<article>`: Represents a self-contained composition, such as a blog post, news article, or forum post, which can be independently distributed or reused.
- `<section>`: Defines a thematic grouping of content, typically with a heading, such as a section of an article.
- `<footer>`: Used to define the footer for a section or a page, typically containing copyright information or links to privacy policies.

```
<footer>
  <p>© 2024 My Website</p>
  <p><a href="#privacy-policy">Privacy Policy</a></p>
</footer>
```

Benefits of Semantic HTML

- **Accessibility**: Screen readers and other assistive technologies rely on semantic tags to help visually impaired users navigate content.
- **SEO**: Search engines prioritize websites that use semantic elements, as they better understand the content's structure.

c. Headings and Paragraphs: Structuring Text Content

Headings and paragraphs form the core structure of text content on web pages.

- **Headings (`<h1>` to `<h6>`)**:
  HTML provides six levels of headings, `<h1>` being the highest and most important, down to `<h6>`. Headings are used to organize content into hierarchical sections, improving both readability and accessibility.

  ```
  <h1>Main Heading</h1>
  <h2>Subheading 1</h2>
  <h3>Subheading 2</h3>
  ```

  > **Note**: There should typically be only one `<h1>` per page, as this represents the primary topic of the page.

- **Paragraphs (`<p>`)**:
  The `<p>` tag is used to define paragraphs of text. It's one of the most common elements you'll use to display blocks of text on your webpage.

  ```
  <p>This is a paragraph that explains a certain point on the webpage.</p>
  ```

  Headings and paragraphs together form the **text structure** of any web page, making it easier for users to navigate and understand content.

d. Creating Links: Navigation with the `<a>` Tag

Links are one of the most important features of the web. They allow users to navigate from one page to another, either within the same website or to external sites.

- **The Anchor Tag (`<a>`)**:
  The `<a>` tag creates a hyperlink, allowing users to click and navigate to another web page or resource. The `href` attribute defines the destination of the link.

  ```
  <a href="https://example.com">Visit Example</a>
  ```

- **Relative vs. Absolute URLs**:

- **Absolute URLs**: Full web addresses that include the protocol (`http://` or `https://`), domain, and path. Used to link to external websites.
- **Relative URLs**: Shortened links that point to resources within the same website.

```
<!-- Absolute URL -->
<a href="https://example.com/about">About Us</a>

<!-- Relative URL -->
<a href="/about">About Us</a>
```

e. Inserting Images: Displaying Visual Content with `<img>`

Images are a vital part of web pages, adding visual appeal and context to the content. The `<img>` tag is used to embed images.

- **The Image Tag (`<img>`)**:
  The `<img>` tag is a self-closing tag that requires two main attributes:
  - `src`: Specifies the location (URL) of the image file.
  - `alt`: Provides alternative text for the image, which is essential for accessibility (e.g., screen readers) and is displayed when the image cannot be loaded.

```
<img src="images/logo.png" alt="Website Logo" />
```

- **Relative vs. Absolute Paths for Images**:
  - **Relative Paths**: Points to images stored within the website's directory.
  - **Absolute Paths**: Full URL to an external image hosted on another website.

```
<!-- Absolute path -->
<img src="https://example.com/images/pic.jpg" alt="External Image" />

<!-- Relative path -->
<img src="images/local-pic.jpg" alt="Local Image" />
```

HTML Basics Key Takeaways

- **HTML Structure**: A well-formed HTML document consists of the `<html>`, `<head>`, and `<body>` tags.
- **Semantic HTML**: Using semantic elements like `<header>`, `<main>`, and `<footer>` improves accessibility and SEO.
- **Headings and Paragraphs**: Headings structure the content into sections, while paragraphs define blocks of text.
- **Links and Images**: The `<a>` and `<img>` tags are essential for creating clickable links and embedding images.