**ALX – FRONT-END PROGRAM**

Web development terminologies

Here are some terminologies that you need to familiarize yourself with as you beginner your web development journey through this course.

**1. HTML (Hypertext Markup Language):**

HTML is the standard markup language used to structure the content of web pages. It provides a set of tags and elements that define the structure and layout of a web page.

**2. CSS (Cascading Style Sheets):**

CSS is a stylesheet language that is used to describe the presentation and styling of HTML documents. It enables you to control the colors, fonts, layouts, and other visual aspects of a web page.

**3. JavaScript:**

JavaScript is a programming language that allows you to add interactivity and dynamic behavior to web pages. It enables you to create interactive forms, perform calculations, manipulate the content on a page, and much more.

**4. Front-end Development:**

Front-end development refers to the process of creating the user-facing part of a website or web application. It involves writing HTML, CSS, and JavaScript code to build the visual and interactive components that users interact with.

**5. Back-end Development:**

Back-end development focuses on the server-side of web development. It involves handling data storage, managing databases, and implementing server-side logic to support the functionality of a web application.

**6. Responsive Design:**

Responsive design is an approach to web development that ensures a website or web application looks and functions well on different devices and screen sizes. It involves using CSS media queries and flexible layouts to adapt the content for optimal viewing on desktops, tablets, and mobile devices.

**7. Version Control:**

Version control is a system that tracks and manages changes to files and code over time. It allows multiple developers to collaborate on a project, keep track of different versions of the code, and revert to previous versions if needed. Git is a popular version control system widely used in web development.

**8. IDE (Integrated Development Environment):**

An IDE is a software application that provides a comprehensive set of tools for coding and developing web applications. It includes features such as code editors, debugging tools, and project management capabilities to streamline the development process.

**9. Wireframing:**

Wireframing is the process of creating a visual representation or blueprint of a web page or application’s structure. It helps in planning the layout, content organization, and user interface before diving into the actual design and development.

**10. Deployment:**

Deployment refers to the process of making a website or web application accessible to users on the internet. It involves configuring servers, setting up hosting environments, and transferring files to a live server so that the website can be accessed globally.

These are just a few of the many terminologies you’ll encounter in web development. As you progress, you’ll become more familiar with these terms and gain a deeper understanding of their significance in building web projects.

**3 important terms that you will hear about often**

**1. Web Page:**

A web page is a single document or file within a website that is displayed in a web browser. It contains a combination of HTML, CSS, and JavaScript code that defines the structure, design, and functionality of the content. Web pages can include text, images, videos, links, forms, and other elements that provide information or allow user interaction.

**2. Website:**

A website is a collection of related *web pages* that are hosted on a web server and can be accessed by users through a web browser. It typically consists of various interconnected pages that share a common domain or URL. Websites can be **static**, providing informational content, or **dynamic**, offering interactive features and functionality.

**3. Web App (Web Application):**

A web app, short for web application, is a type of software application that runs on a web server and is accessed through a web browser. Unlike static web pages, web apps are more interactive and dynamic, offering functionality similar to traditional desktop or mobile applications. They can perform complex tasks, process user input, store data, and provide a customized user experience.

Web apps often rely on a combination of front-end technologies (HTML, CSS, JavaScript) and back-end technologies (server-side programming languages, databases) to handle user interactions, data processing, and server communication. Examples of web apps include online banking systems, social media platforms, e-commerce websites, and productivity tools.

These definitions provide a basic understanding of the concepts of websites, web pages, and web apps. As you delve further into web development, you’ll explore more advanced topics and gain a deeper understanding of how these components work together to create engaging and functional web experiences.

# Tools for Web Development: Getting Started

# 1. Web Browser

A web browser is the primary tool for viewing and interacting with websites. Popular web browsers include Chrome, Firefox, Safari, and Edge. You may already have a web browser installed on your computer. If not, you can download and install one of these browsers by visiting their official websites.

## **2. Text Editor or Integrated Development Environment (IDE)**

A text editor or IDE is where you will write and edit your web development code. There are many options available, both free and paid. Some popular choices for text editors include Visual Studio Code, Sublime Text, and Atom.

To install a text editor or IDE, simply visit the official website of your chosen tool and download the installation file. Follow the instructions provided to complete the installation process.

For this course, we recommend the use of Visual studio code. Find below details on how to install Visual studio code

**3. Version Control System**

Version control is essential for tracking changes to your code and collaborating with others. Git is the most widely used version control system in the web development community.

**4. Command Line Interface (CLI)**

A command line interface (CLI) provides a text-based way to interact with your computer and execute various commands. It is often used for running development tools, managing project dependencies, and automating tasks.

Depending on your operating system, you may already have a built-in command line interface (such as Terminal on macOS or Command Prompt on Windows). Alternatively, you can use a more feature-rich terminal emulator like iTerm2 or PowerShell.

By installing these essential tools, you will have a solid foundation to start your web development journey. Remember to consult the documentation and resources provided by each tool to fully understand their features and capabilities. Happy coding!

# Introduction to the Web

**What is Web Development?**

Web development is the process of creating websites and web applications that are accessible on the internet. It involves combining various technologies, such as HTML, CSS, and JavaScript, to design and build interactive and visually appealing web pages. Web development empowers you to bring your ideas to life, communicate with users, and showcase your creativity through digital platforms.

### Why Learn Web Development?

Learning web development opens up a world of possibilities. Here are three reasons why it’s worth diving into this field:

#### 1. Creative Expression:

Web development allows you to unleash your creativity and design visually captivating and engaging websites. You can bring your unique ideas to life, experiment with different layouts, colors, and interactive elements to craft remarkable online experiences.

#### 2. In-Demand Skills:

In today’s digital age, businesses and organizations rely heavily on their online presence. By acquiring web development skills, you become a sought-after professional who can contribute to creating and maintaining websites, web applications, and e-commerce platforms.

#### 3. Career Opportunities:

Web development offers a broad range of career opportunities. You can work as a front-end developer, focusing on user interfaces and client-side interactions, or as a back-end developer, handling server-side logic and database management. Additionally, you can explore freelance work, start your own web development business, or collaborate with teams on exciting projects.

**WEEK -01**

**GIT and GIT HUB**

Working with Repositories on GitHub

Introduction to GitHub Repositories

Repositories, or “repos,” are at the heart of GitHub. They are essentially directories where all the files for your project are stored, including the revision history of each file. Repositories can be public or private and can include various resources such as code, documentation, images, and more. Understanding how to interact with repositories is fundamental to leveraging GitHub’s full potential.

Forking a Repository

**What is Forking?**

Forking is the process of creating a personal copy of someone else’s project. This allows you to freely experiment with changes without affecting the original project. It’s also the first step in contributing to open source projects or when you want to use someone else’s project as a starting point for your own.

**How to Fork a Repository:**

1. **Find the Repository:** Navigate to the GitHub page of the repository you wish to fork.
2. **Fork the Repository:** Click on the “Fork” button, usually located at the top right of the page. GitHub will then create a copy of the repository in your account.
3. **Clone Your Fork (optional):** To work on your fork locally, you can clone it to your computer, which we’ll cover next.

Cloning a Repository from GitHub

**What is Cloning?**

Cloning is the process of making a **local** copy of a GitHub repository on your computer. You can think of it as **downloading** the repository from GitHub onto your own computer. This allows you to work on the project locally, make changes, and push updates back to GitHub.

Practically what this means is that after downloading the repository onto your computer, you can now work on that code (edit or add new code) and once you are satisfied with the changes, you will need to send the updated code back to GitHub so that others can also see the changes you have made or you yourself will always have the most up to date code when you check your GitHub.

**How to Clone a Repository:**

1. **Copy the Repository URL:** On the GitHub page of the repository (it could be the original repository or your fork), click the “Code” button and copy the URL shown under “Clone with HTTPS” or “Clone with SSH.”
2. **Clone the Repository:**
   * Open your terminal or Git Bash.
   * Navigate to the directory where you want the repository to be copied.
   * Use the git clone command followed by the copied URL: git clone https://github.com/username/repository-name.git
   * Replace https://github.com/username/repository-name.git with the actual URL you copied.

**After Cloning:**

Once the repository is cloned, you can navigate into the directory that Git created (which will have the same name as the repository) using the cd command. From here, you’re ready to start working on the project.

Practical Example for you to try out

1. **Fork a Repository:** Find an open-source project on GitHub that interests you and fork it to your account.
2. **Clone Your Fork:** Clone the forked repository to your local machine and explore its contents.
3. **Make a Small Change:** As a simple exercise, try adding a text file with your name or a small note. Commit this change locally.
4. **Reflect:** Reflect on the process of forking and cloning. Consider the possibilities this opens for collaboration and learning.

Conclusion

Forking and cloning are key features of GitHub that promote collaboration and learning. By forking a repository, you can make independent changes without affecting the original project. Cloning allows you to work on a project locally, giving you the flexibility to experiment and develop. Together, these tools empower you to engage with the global developer community, contribute to projects, and kickstart your own initiatives.

**Integrating Git with GitHub**

Creating a New Repository on GitHub

**Step 1: Create a New Repository**

1. **Log in** to your GitHub account.
2. Click the **+** icon in the top-right corner and select **New repository**.
3. Name your repository, add a description (optional), and choose whether it will be public or private.
4. Initialize the repository with a README file to get started.
5. Click **Create repository**.

Cloning Your Repository

**Step 2: Clone the Repository to Your Local Machine**

1. After creating your repository on GitHub, click the **Code** button and copy the URL under “Clone with HTTPS” or “Clone with SSH”.
2. Open your terminal or Git Bash.
3. Navigate to where you want your repository to be on your local machine using the cd command.
4. Clone the repository using: git clone <REPOSITORY-URL>
5. Enter your GitHub credentials if prompted.

Making Changes and Committing

**Step 3: Work Locally on Your Project**

1. Navigate into your repository’s directory: cd <REPOSITORY-NAME>
2. Make changes or add new files to your project.
3. Stage your changes: git add .
4. Commit your changes with a meaningful message: git commit -m "Your commit message"

Pushing Changes to GitHub

**Step 4: Push Your Changes**

1. Ensure you are on the branch you wish to push. If it’s the main branch: git checkout main
2. Push your changes to GitHub: git push origin main

Working with Branches

**Step 5: Creating and Pushing Branches**

1. Create a new branch for a feature or bugfix: git checkout -b <BRANCH-NAME>
2. Make changes, stage, and commit them as shown above.
3. Push the new branch to GitHub: git push origin <BRANCH-NAME>

Collaborating Through Pull Requests

**Step 6: Merge Changes via Pull Requests**

1. On GitHub, navigate to your repository.
2. Click the **Pull requests** tab and then click **New pull request**.
3. Select the branch you pushed from the “compare” dropdown.
4. Review your changes and click **Create pull request**.
5. Add a title and description for your pull request.
6. Once the pull request is reviewed by collaborators, it can be merged into the main branch.

Conclusion

By integrating Git with GitHub, you harness a powerful toolkit for version control and collaboration. This workflow enables you to manage projects efficiently, contribute to open source, and work effectively in teams. Remember, the key to mastering Git and GitHub lies in practice and participation in projects that interest you.

# Collaboration on GitHub

### Introduction to Collaboration

Collaboration is a cornerstone of GitHub, enabling developers from around the globe to work together on projects. At the heart of this collaborative effort are pull requests and code reviews, processes that allow for shared development and refinement of code in a structured and efficient manner.

#### What are Pull Requests?

Pull requests are GitHub’s way of letting you tell others about changes you’ve pushed to a branch in a repository. Once a pull request is opened, you can discuss and review the potential changes with collaborators, make further commits if necessary, and eventually merge the pull request into the main branch of the project.

##### **Creating a Pull Request**

1. **Push your branch to GitHub:** Before you can create a pull request, you need to have a branch with changes that have been pushed to your repository on GitHub.
2. **Start a new pull request:** Navigate to your repository on GitHub and click the “Pull requests” tab, then click the “New pull request” button.
3. **Choose the branches:** Select the base branch (the one you want the changes to be merged into) and the compare branch (the one with your changes).
4. **Review changes and create the pull request:** Look over the differences shown, enter a title and description for your pull request, and then click “Create pull request.”

#### Reviewing Code and Merging Pull Requests

Once a pull request is created, collaborators can review the changes, discuss modifications, and even push follow-up commits if necessary.

##### **Reviewing Code**

1. **Navigate to the pull request:** Click on the “Pull requests” tab in your repository and select the pull request you want to review.
2. **Add comments:** Use the “Files changed” tab to view the proposed changes. You can add comments to specific lines of code or provide general feedback in the conversation tab.
3. **Approve or request changes:** After reviewing, you can approve the changes, request further modifications, or simply leave comments without a formal approval.

##### **Merging Pull Requests**

Once the pull request has been reviewed and approved, it can be merged into the base branch.

1. **Click “Merge pull request”:** This button is available once the pull request has been approved or if you have write access to the repository.
2. **Confirm the merge:** You may choose to modify the commit message or just use the default message that includes a reference to the pull request number.
3. **Delete the branch (optional):** After merging, you have the option to delete the branch used for the pull request, keeping the repository tidy.

### Best Practices for Collaboration

* **Communicate clearly:** Use descriptive titles and detailed descriptions in your pull requests.
* **Keep pull requests focused:** Limit pull requests to a single feature or bug fix when possible.
* **Respond to feedback:** Be open to and respectful of feedback from collaborators during the review process.

### Conclusion

**Pull request**s and code reviews are powerful tools for collaboration on GitHub, allowing teams to discuss changes, improve code quality, and contribute to the success of a project. By understanding and utilizing these tools, you can effectively collaborate with others, whether you’re working on small projects or contributing to large open-source repositories.

Basic Git Commands

Understanding the basic commands in Git is essential for tracking changes in your projects and collaborating with others. This section will guide you through creating and managing your repositories with some fundamental Git commands.

**Creating a New Repository (init)**

**A repository (or “repo”) is where all the files for a particular project are stored**, along with the revision history.

1. **Open your terminal** and navigate to the directory where you want to create a new repository.
2. **Run the git init command**: git init
3. This command creates a new subdirectory named .git that houses all of your repository’s configuration files and history. Your current directory is now a Git repository, ready to track project files.

Cloning an Existing Repository

**Cloning is the process of creating a local copy of a remote repository.**

1. **Find the repository** you wish to clone. This could be a project on GitHub or another Git server. Copy the repository’s URL.
2. **In your terminal**, navigate to the directory where you want to clone the repository.
3. **Run the git clone command** followed by the copied URL: git clone https://github.com/exampleuser/example-repo.git
4. This command copies all the data from the repository to your local machine, allowing you to work on the project.

Adding and Committing Changes

After making changes to your project, you’ll want to save these changes to your **repository**. This process involves two steps: adding changes and committing them.

Adding Changes

1. **With changes made** to your project files, open your terminal and navigate to your repository’s directory.
2. **Run the git add command** to stage changes for commit. To add a specific file: git add filename.txt Or, to add all changed files**: git add .**

Committing Changes

1. **After adding your changes**, it’s time to commit them. Use the git commit command with a message describing what you’ve done: git commit -m "Your commit message"
2. This command saves your staged changes along with a descriptive message, which helps others understand what you’ve done and why.

Viewing the Commit History (log)

To see the history of your commits, including what changes were made and by whom:

1. **In your terminal**, make sure you’re in your repository’s directory.
2. **Run the git log command**: git log
3. You’ll see a list of commits, each with an author, date, and message. For a more condensed view, try git log --oneline.

Conclusion

Congratulations! You’ve now learned how to create a new Git repository, clone an existing one, stage and commit changes, and view the commit history. These are the foundational skills you need to start tracking your projects with Git.

**Welcome to the "Branching and Merging with Git" Project**

Introduction

Welcome to the next step in your journey with Git! In this project, you will dive into the powerful features of branching and merging. These are essential skills for any developer, allowing you to work on different features, fix bugs, or experiment with new ideas in parallel without affecting the main project.

Project Overview

This project is divided into two main parts, spread over 5 hours of engaging and hands-on learning experiences:

Concept Page 1: Understanding Branches (1 hour)

* **What are Branches?**: Discover the concept of branches in Git, why they are crucial for development workflows, and how they can be used to work on different parts of your project simultaneously.
* **Creating Branches**: Learn how to create new branches in Git, enabling you to safely experiment and make changes to your project.
* **Switching Between Branches**: Master the skill of navigating between different branches within your project, allowing you to manage multiple development streams effortlessly.

Concept Page 2: Merging and Conflicts (2 hours)

* **Merging Branches**: Understand how to merge branches, integrating changes from one branch into another. This process is key to collaborative development and feature integration.
* **Handling Merge Conflicts**: Learn how to identify and resolve merge conflicts, a critical skill for any team member in a collaborative project environment.

Learning Objectives

By the end of this project, you will be able to:

1. Understand the purpose and function of branches in Git.

2. Create and manage branches, seamlessly switching between them.

3. Merge branches to integrate changes, understanding the mechanics behind the process.

4. Identify and resolve merge conflicts, ensuring a smooth development process.

Tools and Requirements

* **Git**: Ensure your Git installation is up to date and configured.
* **Text Editor**: Have a text editor (such as VSCode, Sublime Text, or Atom) ready for editing and managing files.

**Let’s Get Started!**

Branching and merging are at the heart of every collaborative project. They allow teams to work in parallel, experiment, and bring together complex features into a cohesive product. This project will equip you with the knowledge and skills to leverage these powerful Git features confidently.

Prepare to embark on this exciting learning adventure, where you’ll not only understand the theoretical aspects of branching and merging but also apply them through hands-on exercises and practical challenges.

Happy Coding!

**Understanding Branches in Git**

Introduction to Branches

Branches in Git are fundamentally pointers to a specific snapshot of your changes. When you create a project in Git, it automatically creates a main branch (often called master or main). Think of branches as parallel universes where you can work on different features, bug fixes, or experiments without affecting the main body of work. This approach enables multiple developers to work on the same project simultaneously, safely isolating their changes until they’re ready to merge back into the main project flow.

What are branches?

* **Flexibility**: Branches offer a way to work on changes, updates, or new features in isolation from the main project, improving workflow flexibility.
* **Collaboration**: They facilitate parallel development among team members, allowing for diverse tasks to be completed simultaneously without interference.
* **Experimentation**: Developers can experiment with new ideas in a branch without risking the stability of the main project.

Creating Branches

Creating a new branch in Git is a simple and instantaneous operation. Here’s how to do it:

1. **Open your terminal** and navigate to your Git project directory.
2. **Use the git branch command** to create a new branch. Replace <branch-name> with a meaningful name for your branch: git branch <branch-name>
3. You’ve now created a new branch, but you’re still in your current branch. Git doesn’t automatically switch to the new branch when you create it.

Best Practices for Naming Branches

* Use short, descriptive names.
* Incorporate task identifiers if linked to a specific task or issue.
* Separate words with hyphens to improve readability.

Switching Between Branches

To move from one branch to another and bring your working directory up to date with the selected branch’s latest commits, you use the git checkout command.

1. **To switch to an existing branch**, enter the following command, replacing <branch-name> with the name of the branch you want to switch to: git checkout <branch-name>
2. If you want to create a new branch and immediately switch to it, you can use: git checkout -b <new-branch-name> This command combines the creation and switching of branches into one step.

Practical Example for you

Try creating and switching between branches in a sample project:

1. Create a new branch named feature-x and switch to it.
2. Make some changes in your project under this branch (e.g., create a new file or edit an existing one).
3. Switch back to your main branch.
4. Reflect on how the changes made in feature-x are isolated from the main branch.

Conclusion

**Branches** are a powerful feature of Git that enable efficient, parallel development workflows. By creating branches, you can work on different aspects of a project simultaneously without impacting the main line of development. Remember, the goal of using branches is to keep your project organized, facilitate collaboration, and allow for safe experimentation.

Happy branching!

**Merging Branches and Handling Conflicts in Git**

*Introduction to Merging*

**Merging** is a fundamental Git operation that allows you to combine the history of separate branches into one. In most workflows, merging is used to integrate the work completed in a feature branch back into the main branch once development is complete. Understanding how to merge branches is crucial for maintaining a coherent and unified project history.

What is Merging?

* **Integration**: Merging takes the changes from one branch (source) and integrates them into another (target), typically the main branch.
* **Non-Destructive Operation**: The source branch’s history remains intact after the merge, allowing for continued development or preservation.

How to Merge Branches

1. **Ensure you’re on the target branch** where you want to integrate the changes. If you’re merging into main, you should switch to main: git checkout main
2. **Merge the source branch** into your current branch (e.g., merging feature-x into main): git merge feature-x
3. If the merge is successful without conflicts, Git will auto-create a merge commit if necessary, combining the histories of the merged branches.

Handling Merge Conflicts

Not all merges go smoothly. Sometimes, Git can’t automatically reconcile differences between the two branches, leading to merge conflicts. Conflicts typically occur when the same part of a file has been differently modified in both branches.

Identifying Merge Conflicts

When Git encounters a conflict during a merge, it will pause the operation, allowing you to resolve the conflict. Git modifies the affected files to visually show the conflicting changes, marking the start and end of each section with <<<<<<<, =======, and >>>>>>>.

Resolving Merge Conflicts

When Git cannot automatically merge changes from different branches due to conflicting modifications, it halts the merge process, requiring manual intervention to resolve the conflicts. Here’s how to navigate this process effectively:

Step 1: Identifying Conflicts

After attempting a merge that results in conflicts, Git will notify you which files are affected. Begin by opening these files in your text editor or IDE to locate the conflict markers.

Step 2: Understanding Conflict Markers

Within the conflicted file(s), Git uses specific markers to indicate the conflicting sections:

* <<<<<<< HEAD: Marks the beginning of the conflicting changes from the current branch (the one you’re merging into).
* =======: Divides your changes from the changes in the other branch.
* >>>>>>> [other branch name]: Marks the end of the conflicting changes from the other branch (the one you’re merging from).

Step 3: Resolving the Conflict

1. **Review the Conflicts**: Carefully examine the code between the conflict markers. Understand the changes from both branches and decide how to integrate them.
2. **Edit the File**: Make the necessary modifications to resolve the conflict. This may involve choosing one set of changes over the other, merging the changes manually, or even creating a new solution that incorporates elements of both.
3. **Remove the Conflict Markers**: After you’ve edited the file to your satisfaction, delete the <<<<<<<, =======, and >>>>>>> markers. The file should now look exactly how you want it after the merge.
4. **Test Your Changes**: Before finalizing the merge, it’s crucial to test your code to ensure that your resolutions didn’t introduce any errors.

Step 4: Finalizing the Resolution

1. **Add the Resolved Files**: Once you’re confident in your resolution, stage the changes for commit by using: git add <file-name> Repeat this for each file you’ve resolved.
2. **Complete the Merge**: With all conflicts resolved and the changes staged, complete the merge by committing: git commit Git will prompt you to enter a commit message for the merge. The default message typically includes a brief note about the conflict resolution. You can edit this message to add any additional context if necessary.
3. **Verify the Merge**: Use git log or other tools to ensure that the merge commit looks correct and all branches have been integrated as intended.

Practical Example for you to try out

* **Create a Conflict**: On two separate branches, make different changes to the same part of the same file.
* **Attempt to Merge**: Merge one branch into the other and observe the conflict.
* **Resolve the Conflict**: Follow the steps to resolve the conflict, then complete the merge.

Best Practices for Conflict Resolution

* **Communicate**: If you’re unsure about how to resolve a conflict, discuss it with your team. Sometimes, the original code author can provide valuable insight.
* **Stay Updated**: Regularly pull changes from the main branch into your feature branch to minimize conflicts.
* **Use GUI Tools**: If you’re uncomfortable resolving conflicts from the command line, consider using a graphical tool or IDE that provides a more visual interface for conflict resolution.

Conclusion

**Merging** is a powerful feature that facilitates the integration of separate development efforts into a cohesive project. While merge conflicts can be a common occurrence in collaborative environments, understanding how to resolve them ensures smooth project progress. By mastering merging and conflict resolution, you’ll enhance your ability to contribute effectively to any Git-managed project.

Happy merging!

Issues and Labels on GitHub

Introduction to GitHub Issues

GitHub Issues are a powerful tool for tracking tasks, enhancements, bugs, and other types of work within a project. They serve as a centralized forum for discussing project-related topics, allowing team members to collaborate more effectively.

Using Issues for Project Management

**Creating an Issue:**

1. **Navigate to your repository** on GitHub and click on the “Issues” tab.
2. Click the **New issue** button.
3. Provide a **title** and **description** for the issue. Be as specific as possible to help team members understand the task or problem.
4. **Assign** the issue to a team member, set a **milestone**, or add **labels** as needed.
5. Click **Submit new issue**.

**Managing Issues:**

* **Assignees**: Assign issues to team members responsible for working on them.
* **Milestones**: Group issues into milestones to track progress towards specific project goals.
* **Closing Issues**: Once an issue is resolved, close it to keep the project board tidy. You can reference the commit that resolved the issue by including the issue number in the commit message (e.g., “Fixes #123”).

Tagging Issues and Pull Requests with Labels

Labels are a versatile tool for categorizing issues and pull requests. They can indicate the status, priority, or type of work (bug, feature request, etc.).

Creating and Applying Labels

**Creating a Label:**

1. In your repository, click on the “Issues” tab, then find the “Labels” link near the top.
2. Click **New label**.
3. Name your label, choose a color, and provide a brief description.
4. Click **Create label**.

**Applying Labels:**

* When creating or editing an issue or pull request, click on the **Labels** section on the right side and select the labels you want to apply.
* You can apply multiple labels to a single issue or pull request to provide more context.

Best Practices for Using Labels

* **Consistency**: Use a consistent labeling system across your project to make it easy for team members to understand what each label means.
* **Prioritization**: Use labels to indicate priority levels (e.g., “high priority,” “low priority”) to help the team focus on important tasks.
* **Categorization**: Create labels for different types of work (e.g., “bug,” “feature”) and aspects of the project (e.g., “documentation,” “UI/UX”) to simplify organization and searching.

Conclusion

GitHub Issues and Labels are essential tools for effective project management, offering a structured way to track and organize work within a project. By effectively using issues to manage tasks and labels to categorize them, teams can improve collaboration, prioritize work, and achieve project goals more efficiently.

Advanced Git Commands

Welcome to the exploration of Advanced Git Commands, designed to enhance your workflow and manage your projects more efficiently. In this section, we’ll delve into the concepts of stashing changes and utilizing .gitignore to streamline your development process.

Stashing Changes

When you’re in the middle of work and need to switch branches to work on something else, but you’re not ready to commit the current changes, Git stash is your friend. It temporarily shelves (or stashes) changes so you can work on a different task.

**How to Use Git Stash:**

1. **Stashing Your Work:** To stash changes, simply run:

git stash

This command takes your modified tracked files and staged changes and saves them for later use, reverting them to their original state.

1. **Listing Stashes:** You can have multiple stashes. To see all your stashed changes, use:

git stash list

1. **Applying a Stash:** To apply the most recently stashed changes and keep the stash for potential later use, run:

**git stash apply**

If you want to apply a specific stash, use:

**git stash apply stash@{<stash\_number>}**

To apply the most recent stash and remove it from the stash list, run:

git stash pop

1. **Removing Stashes:** To clear your stashes if you no longer need them, you can run:

git stash clear

Or to drop a specific stash, use:

git stash drop stash@{<stash\_number>}

Using .gitignore

The .gitignore file is a powerful tool that tells Git which files or directories to ignore in a project. It’s particularly useful for excluding files that don’t need to be part of your repository, such as temporary files, build artifacts, or sensitive information.

**How to Use .gitignore:**

1. **Creating a .gitignore File:**
   * Create a file named .gitignore in your project’s root directory.
   * You can specify patterns to ignore. For example:
2. # Ignore all txt files
3. \*.txt
4. # But track this one specifically
5. !important.txt
6. # Ignore all files in the build/ directory
7. build/
   * Files and directories matching these patterns will be ignored by Git.
8. **Common .gitignore Configurations:**
   * Many programming languages and development environments have common files and directories that should be ignored. You can find templates for .gitignore files tailored to various languages and tools on GitHub’s [gitignore repository](https://intranet.alxswe.com/rltoken/31-bCFlSkbBWG13nXtUCRQ" \o "gitignore repository" \t "_blank).
9. **Updating .gitignore:**
   * If you add a pattern to .gitignore for files that were already tracked by Git, those files will continue to be tracked. To stop tracking them, you must explicitly remove them from the repository:
10. git rm --cached <file>
    * Commit your changes after updating .gitignore or removing files.

Conclusion

Mastering advanced Git commands like stashing changes and utilizing .gitignore significantly improves your efficiency and project management capabilities.

Stashes provide a flexible way to switch contexts without losing progress, while .gitignore helps maintain a clean and relevant repository. Together, these tools empower you to manage your development workflow more effectively.

**TASK**

0. Initialize a Local Repository and Push to GitHub

**mandatory**

**Objective:** Learn how to create a file locally, initialize a Git repository, commit your changes, and push the repository to GitHub.

**Skills Practiced:**

* Initializing a Git repository
* Basic Git commands (add, commit, push)
* Creating a new repository on GitHub
* Pushing a local repository to GitHub

**Task Steps:**

1. **Create a New File Locally:**
   * On your computer, create a new directory named LocalRepoProject.
   * Inside this directory, create a file named hello.txt.
   * Open hello.txt in a text editor and add the following content:
2. Hello, Git and GitHub!
3. **Initialize a Git Repository:**
   * Open a terminal or command prompt.
   * Navigate to the LocalRepoProject directory.
   * Run the following command to initialize a new Git repository:
4. git init
5. **Commit Your File to the Repository:**
   * Add hello.txt to the staging area with the command:
6. git add hello.txt
   * Commit the file to your repository: git commit -m "Add hello.txt with greeting"
7. **Create a New Repository on GitHub:**
   * Log in to your GitHub account.
   * Click the **New** button (or navigate to [this link](https://intranet.alxswe.com/rltoken/jFWrHc_B7wgbjRHYQkEfrA" \o "this link" \t "_blank)) to create a new repository.
   * Name your repository LocalRepoProject. Do not initialize the repository with a README, .gitignore, or license.
   * Click **Create repository**.
8. **Push the Local Repository to GitHub:**
   * After creating the repository on GitHub, you’ll be shown a URL for the repository. Copy this URL.
   * Back in your terminal, link your local repository to GitHub with the command:
9. git remote add origin <REPOSITORY-URL>

* Push your commits to GitHub:
* git push -u origin main

**Expected Outcome:**

You will have a local Git repository with a single file (hello.txt) committed. This repository will be pushed to GitHub, where the file will be visible in your new LocalRepoProjectrepository.

**Submission:**

We have access to your GitHub account so the checker will be able to directly check your submissions. Ensure your repository is public so it can be reviewed by the checker.

**Notes:**

* This project is designed to introduce the fundamental workflow of using Git locally and pushing changes to a remote repository on GitHub.
* Ensure you replace <REPOSITORY-URL> with the actual URL provided by GitHub when you create your online repository.

**Repo:**

* GitHub repository: LocalRepoProject
* File: hello.txt

1. Clone a GitHub Repository and Add New Content

**mandatory**

**Objective:** Familiarize yourself with the process of cloning an existing repository from GitHub, adding new content to it, and pushing the changes back to GitHub.

**Skills Practiced:**

* Cloning a repository from GitHub
* Editing files and committing changes in a Git-enabled directory
* Pushing updates to a remote repository

**Task Steps:**

1. **Create a New Repository on GitHub:**
   * Log in to your GitHub account.
   * Navigate to the repository creation page (https://github.com/new).
   * Name your repository GitHubCloneProject. Optionally, add a description.
   * Choose the visibility as public.
   * Click **Create repository**.
2. **Clone the Repository:**
   * Once the repository is created, copy the HTTPS link as shown in the above image.
   * Open your terminal or command prompt.
   * Choose a suitable directory on your local machine where you want to clone the repository.
   * Execute the clone command:
3. git clone <REPOSITORY-URL>

* Replace <REPOSITORY-URL> with the URL you copied from GitHub.

1. **Add New Content:**
   * Navigate into the cloned repository’s directory:
2. cd GitHubCloneProject

* Create a new file named greetings.txt.
* Edit greetings.txt in your text editor and add the following content:
* Hello, this is a message from my cloned repository!
  + Create a new file called README.md and put the text This is the description of my repo into it.

1. **Commit and Push Your Changes:**
   * Add the new file to the staging area:
2. git add .

* Commit the new file to your local repository:
* git commit -m "Add greetings.txt and README.md"
* Push your changes to the GitHub repository:
* git push origin master

**Expected Outcome:**

Your GitHub repository GitHubCloneProject will now include a greetings.txt file with the specified content. This demonstrates your ability to clone a repository, make changes locally, and push those changes back to the remote repository.

**Notes:**

* This project reinforces the concept of remote repositories and introduces you to managing and updating project content after cloning it from GitHub.
* Remember to replace <REPOSITORY-URL> with your repository’s actual URL provided by GitHub.

**Repo:**

* GitHub repository: GitHubCloneProject
* File: greetings.txt, README.md

2. Branching, Making Changes, and Resolving Merge Conflicts

**mandatory**

**Objective:** Practice creating branches for feature development, merging branches, and resolving merge conflicts to simulate a common collaborative workflow on Git and GitHub.

**Skills Practiced:**

* Branching and navigating between branches
* Making changes in different branches and committing those changes
* Merging branches and resolving conflicts

**Task Steps:**

1. **Initial Setup:**
   * Use the GitHubCloneProject repository created in the previous task. Create a README.md file if you don’t already have one.
   * Make sure you have some content in the README.md. You can put in the README file the objective of this task.
   * Clone the repository to your local machine if you haven’t already:
2. git clone <REPOSITORY-URL>

* Navigate into the repository’s directory:
* cd GitHubCloneProject

1. **Create a New Branch for a Feature:**
   * Create and switch to a new branch named feature-greeting:
2. git checkout -b feature-greeting

* Create a new file named feature.txt and add some content related to a feature, for example:
* This is a new feature.
* Add and commit the change:
* git add feature.txt
* git commit -m "Add feature description"

1. **Make Changes in the Main Branch:**
   * Switch back to the main branch:
2. git checkout main

* Modify the README.md file by adding a new line at the bottom:
* Modification in main branch.
* Commit the change:
* git add README.md
* git commit -m "Modify README in main branch"

1. **Merge the Feature Branch and Resolve a Conflict:**
   * Attempt to merge feature-greeting into main:
2. git merge feature-greeting

* If a merge conflict occurs, open the conflicting file(s) in a text editor. Manually resolve the conflicts by editing the file to keep or integrate both sets of changes.
* After resolving the conflict, add the file to the staging area and commit the merge:
* git add .
* git commit -m "Resolve merge conflict between main and feature-greeting"

1. **Push Changes to GitHub:**

* Push the updated main branch, including the merged feature and resolved conflict, to GitHub:
* git push origin main

**Expected Outcome:**

You will have successfully created a new feature branch, made changes in both the feature-greeting and main branches, and resolved a merge conflict resulting from attempting to merge these branches. Your GitHubCloneProject repository on GitHub will reflect these changes and the merge resolution.

**Notes:**

* This project simulates a collaborative scenario where changes made in different branches can lead to conflicts. It tests your ability to manage branches, navigate merge conflicts, and maintain a clean project history.
* Replace <REPOSITORY-URL> with your repository’s actual URL.

**Repo:**

* GitHub repository: GitHubCloneProject
* File: feature.txt, README.md

**WEEK 02**

# UX/UI DESIGNS

**Introduction**

UI (User Interface) design involves creating the visual elements and interactive features that users experience when interacting with a digital product. UI design tools are specialized software applications that streamline this process, enabling designers to:

* Conceptualize and brainstorm ideas.
* Design user interfaces with visual elements and layout.
* Prototype interactive experiences to test usability.
* Handoff designs to developers for implementation.

**Exploring UI Design Tools**

The UI design landscape offers a diverse range of software options, each with its own strengths and functionalities. Here are some of the most popular tools:

Design Tools

**Figma**: A web-based design platform known for its collaborative features, ease of use, and robust prototyping capabilities. [Figma](https://intranet.alxswe.com/rltoken/N6j-824ncnf7swhocTJkPA" \o "Figma" \t "_blank)

**Sketch**: A powerful vector graphics editor specifically designed for UI and UX design, offering a vast plugin ecosystem for customization. (Note: Currently macOS only) [Sketch](https://intranet.alxswe.com/rltoken/8MetQWrAYmwYXVK8LisBfg" \o "Sketch" \t "_blank)

**Adobe XD**: Part of the Adobe Creative Suite, XD provides a comprehensive solution for UI design, prototyping, and animation. [AdobeXD](https://intranet.alxswe.com/rltoken/IJbyAbxt_ftx0iDvDHCBzA" \o "AdobeXD" \t "_blank)

**Prototyping Tools**

**InVision**: A cloud-based platform for creating interactive prototypes with animations and user flows, facilitating user testing and feedback. [Invision](https://intranet.alxswe.com/rltoken/RmzW22gsgwaaqaCv5BzVwA" \o "Invision" \t "_blank)

**Proto.io**: Another popular prototyping tool offering high-fidelity prototyping capabilities and integrations with design software. (<https://en.wikipedia.org/wiki/Proto.io>)

**Other Useful Tools**

**Wireframing Tools**: Tools like Balsamiq or Mockplus allow for quick and low-fidelity wireframing to establish basic layouts before diving into detailed design. [Balsamiq](https://intranet.alxswe.com/rltoken/4jiAwZFOjlKpKIFiQ3qjYA" \o "Balsamiq" \t "_blank)

**Design System Tools**: Tools like Zeplin or Avocode help manage design systems, ensuring consistency across design teams and projects. [Zeplin](https://intranet.alxswe.com/rltoken/MhwftT3DeiKG1Qh3dNnTnQ" \o "Zeplin" \t "_blank)

**Stages of the UI Design Process with Software Support**

**Ideation & Brainstorming**: Tools like Figma or Miro (a collaborative whiteboard tool) can be used to create mood boards, mind maps, and sketches to explore initial design concepts. [miro](https://intranet.alxswe.com/rltoken/clT8vGdebTKa_yoPL3t8fA" \o "miro" \t "_blank)

**Design & Visuals**: Design tools like Figma or Sketch allow creating high-fidelity mockups with customizable UI elements, icons, and typography.

**Prototyping & User Testing**: Prototyping tools like InVision or Proto.io enable the creation of interactive prototypes that simulate user interactions and test design usability.

**Handoff & Development**: Design tools often offer features to export assets and style guides (documentation for developers) to ensure design fidelity during implementation.

**SOME BEST DESING RESOURCES**

* + 100 DailyUi.webflow.oi
  + Icon8.com
  + **Invision App \*\***
  + Uplabs
  + **Figma \*\***
  + **Adobe Xd**
  + Undraw
  + **Sketchapp sources\*\***
  + Paaaterns
  + Drawkit
  + Manypixels
  + Humaaans
  + Avaaaaters
  + Free Illustration
  + Freebies Sketch
  + Blush
  + IRA Designs
  + UI Design Daily
  + UI Space
  + Opendoodles.
  + **Marvel**

### Design Thinking in UI/UX Design

Design Thinking is a cyclical, user-centered process that helps designers solve complex problems and create innovative solutions. It emphasizes understanding the user’s needs, brainstorming ideas, and testing prototypes to ensure the final product meets those needs.

## **Detailed Explanation**

**#### Introduction to Design Thinking**

Design Thinking is a human-centered approach to problem-solving that is widely used in various fields, including UI/UX (User Interface/User Experience) design. It involves understanding the needs and desires of users, challenging assumptions, and creating innovative solutions that truly meet those needs.

#### Design Thinking Process for UI/UX Design

Design Thinking is an iterative process, meaning that it’s not linear, and you may cycle through the stages multiple times as you refine your design. It emphasizes a user-centric approach, focusing on understanding and addressing the needs of the users throughout the design process. Here are the fundamental stages:

* **Empathize**: This is the first stage of the Design Thinking process. It involves understanding the users for whom you are designing the product. This means observing, engaging, and empathizing with the users to understand their experiences, needs, and motivations. Techniques such as interviews, surveys, and user testing can be used to gather insights.
* **Define**: In this stage, you analyze the information gathered during the empathize stage to define the core problems and challenges faced by the users. This involves synthesizing the data collected and identifying user needs, pain points, and goals. The goal is to create a clear and concise problem statement that will guide the design process.
* **Ideate**: Once you have a clear understanding of the problem, the next step is to generate creative ideas to solve it. This stage encourages brainstorming and thinking outside the box. No idea is too wild or impractical at this stage. Techniques such as *brainstorming* sessions, sketching, and mind mapping are commonly used to generate a wide range of ideas.
* **Prototype**: In this stage, you start to turn your ideas into tangible representations. Prototypes can be low-fidelity sketches, wireframes, or high-fidelity mockups, depending on the complexity of the project. The goal is to create a simplified version of the final product that can be tested and iterated upon.
* **Test**: Once you have a prototype, it’s time to test it with real users. This involves gathering feedback on the prototype to see how well it meets the needs of the users and identifies any usability issues. User testing can be done through interviews, surveys, or usability testing sessions. The feedback gathered during this stage is used to refine and improve the design.
* **Iterate**: Based on the feedback gathered during the testing stage, you make necessary revisions to the prototype. This could involve refining certain features, adding new ones, or completely redesigning parts of the product. The goal is to continuously iterate and improve the design based on user feedback until you have a solution that effectively addresses the problem.

#### Benefits of Design Thinking in UI/UX

* **User-Centered Design**: Design Thinking ensures that the UI/UX is designed around user needs and preferences, leading to higher user satisfaction and engagement.
* **Improved Problem-Solving**: The structured approach helps identify the root cause of user problems and develop effective solutions based on user research.
* **Increased Innovation**: By encouraging creative exploration, Design Thinking fosters innovative UI/UX solutions that can differentiate your product from competitors.
* **Enhanced Collaboration**: The process promotes teamwork and collaboration between designers, developers, and other stakeholders.

# How to create your prototypes

**Introduction**

Prototypes are simulated representations of a product that allow designers and stakeholders to visualize, test, and refine design concepts before investing significant time and resources in development. They are crucial for gathering user feedback, identifying usability issues, and ensuring the final product aligns with user needs.

**Types of Prototypes**

Prototypes come in various forms, each serving a specific purpose in the design process:

**Low-fidelity Prototypes**: These are quick and basic representations, often created with paper and pen. They focus on core functionalities and user flows, ideal for early design exploration and brainstorming.

**Mid-fidelity Prototypes**: More detailed than low-fidelity prototypes, they might involve digital tools for creating wireframes or mockups. They provide a clearer idea of layout, navigation, and information architecture.

**High-fidelity Prototypes**: These are highly realistic representations with some level of interactivity, often created using specialized prototyping software. They simulate the final product’s look, feel, and behavior for in-depth user testing.

#### The Prototyping Process

The prototyping process in UI/UX design is a crucial stage where ideas and concepts are transformed into tangible representations that can be tested and iterated upon. Here’s an overview of the prototyping process:

1. **Define Goals and Objectives**: Before diving into prototyping, it’s essential to have a clear understanding of the project goals and objectives. What problem are you trying to solve? What are the key features and functionalities of the product? Having a clear goal will guide the prototyping process and ensure that the prototype aligns with the project’s objectives.
2. **Gather Requirements**: Once the goals and objectives are defined, gather requirements from stakeholders and users. What are their expectations and needs for the product? Understanding these requirements will help inform the prototype design and ensure that it meets the needs of the users.
3. **Sketching and Wireframing**: The prototyping process often starts with sketching and wireframing. Sketching involves creating rough, hand-drawn sketches of the user interface layout and design. Wireframing takes these sketches to the next level by creating more refined, digital representations of the interface structure and layout. These sketches and wireframes serve as a low-fidelity prototype to visualize the initial concepts and gather feedback from stakeholders.
4. **Create Low-Fidelity Prototype**: Using the wireframes as a guide, create a low-fidelity prototype using prototyping tools or simple mockup tools. Low-fidelity prototypes focus on basic functionality and layout without detailed design elements. They are quick and easy to create, allowing for rapid iteration and testing of different design ideas.
5. **Iterate and Gather Feedback**: Once the low-fidelity prototype is created, gather feedback from stakeholders and users through usability testing sessions or feedback surveys. Use this feedback to identify areas for improvement and iterate on the prototype. This iterative process helps refine the design and ensures that the final product meets the needs of the users.
6. **Create High-Fidelity Prototype**: After multiple iterations and refinements, create a high-fidelity prototype that includes detailed design elements, such as colors, typography, and imagery. High-fidelity prototypes provide a more realistic representation of the final product and allow stakeholders and users to better visualize the user experience.
7. **Usability Testing**: Test the high-fidelity prototype with real users to identify any usability issues or pain points. Observe how users interact with the prototype and gather feedback on their experience. Use this feedback to make further refinements to the prototype.
8. **Finalize Prototype**: Once the prototype has been thoroughly tested and refined, finalize the design and prepare it for development. Ensure that the prototype accurately reflects the desired user experience and meets the project’s goals and objectives.
9. **Hand off to Development**: Finally, hand off the finalized prototype to the development team for implementation. Provide detailed documentation and assets to guide the development process and ensure that the final product aligns with the prototype design.

Throughout the prototyping process, collaboration and communication are essential. Involve stakeholders, designers, developers, and users in the process to gather diverse perspectives and ensure that the prototype effectively addresses the needs of all stakeholders.

**Importance of User Testing and Iteration**

Usability testing with prototypes is vital for the success of your design. By observing how users interact with the prototype, you can:

* + Identify usability issues early in the design process, saving time and resources later.
  + Gain valuable insights into user behavior and preferences.
  + Ensure the design is intuitive, efficient, and meets user expectations.

Iteration is an essential part of prototyping. Based on user feedback, you continuously refine and iterate on your prototypes, leading to a more user-centered and effective final product.

### Introduction to Low Code Tools

**Introduction**

Traditional application development often involves extensive coding, requiring specialized programming skills. Low-code development platforms (LCDPs) offer a revolutionary approach by enabling the creation of functional applications with minimal coding. These platforms provide visual interfaces, drag-and-drop functionality, and pre-built components that simplify the development process.

**What is Low-Code Development?**

Low-code development platforms provide a user-friendly environment for building applications without the need for in-depth coding knowledge. They offer features like:

Drag-and-drop interfaces for building user interfaces (UI) and workflows.

Pre-built components and integrations for common functionalities.

Configuration options to customize applications without extensive coding.

**Benefits of Low-Code Development**

**Faster Development**: Low-code platforms significantly accelerate the development process compared to traditional coding methods.

**Democratization of Development**: These tools empower non-programmers to contribute to application development, fostering collaboration.

**Reduced Costs**: By minimizing the need for specialized developers, low-code development can lead to cost savings.

**Simplified Maintenance**: Many platforms offer built-in deployment and maintenance features, streamlining the application lifecycle.

**Capabilities and Limitations of Low-Code Platforms**

**Capabilities**:

Building basic to moderately complex applications.

Automating workflows and business processes.

Integrating with existing data sources and systems.

**Limitations**:

**Customization limitations**: Low-code platforms may not offer the same level of customization as traditional coding for highly unique applications.

**Vendor lock-in**: Switching platforms later might be challenging due to proprietary configurations.

**When to Consider Low-Code Development**

Low-code development is ideal for scenarios such as:

\* Building internal business applications with well-defined requirements.

\* Rapid prototyping and proof-of-concept development.

\* Creating custom applications for specific departments or teams.

\* Extending existing functionalities of enterprise systems.

**WEBFLOW.com**

In this project, you will engage in the foundational tasks of UI design, including wireframing, prototyping, and user interface creation. The project is structured to help you apply basic design concepts to practical tasks, progressing from simple wireframes to interactive, high-fidelity prototypes.

### Objectives:

1. **Understand and Apply UI Design Principles**: Gain a practical understanding of wireframing, prototyping, and user interface design. You will apply concepts like design thinking, low-code tools, and design aesthetics in real-world scenarios.
2. **Develop Technical Proficiency**: Use industry-standard tools such as Figma, Adobe XD, and Sketch to create wireframes and prototypes. You will start with basic tools and progress to more advanced functionalities.
3. **Enhance Problem-Solving Skills**: Each task is designed to challenge your problem-solving skills, requiring you to consider both user experience (UX) and user interface (UI) as you design.
4. **Prepare for Real-World Applications**: The deliverables are structured to simulate real-world workflows and outputs, preparing you for professional UI/UX design tasks.

### Requirements:

* **Tools**: Familiarity with at least one of the following - Balsamiq, Figma, Adobe XD, or Sketch. Access to these tools, whether through free trials or educational versions, is essential.
* **Skills**: Basic understanding of design principles and enthusiasm for learning UI/UX design. Prior experience in graphic design is helpful but not required.
* **Delivery**: Completion of all tasks, submission of wireframes, prototypes, and user interfaces through links to digital files or screenshots in Google Docs as specified in each task description.

### Project Tasks:

1. **Create a Wireframe for a Personal Portfolio Website**: Focus on layout and structure using simple wireframing tools. Include essential sections such as About Me, Projects, Contact Form, and Blog.
2. **Convert Low-Fi Wireframe to High-Fi Prototype**: Enhance your initial wireframe by adding interactivity, design elements, and content. Focus on aesthetics and user experience.
3. **Design a User Interface for a Todo List Application**: Create a visually appealing and functional UI for a basic task management app. Emphasize usability, interactivity, and scalability.

### Evaluation Criteria:

* **Clarity and Organization**: Wireframes and prototypes should be well-organized, making it easy for reviewers to understand the layout and user flow.
* **Aesthetic and Functional Design**: Designs should not only be visually appealing but also functional, demonstrating a good balance of UI elements.
* **Adherence to Requirements**: Submissions must meet all task instructions and deliverables as specified.
* **Creativity and Innovation**: While following the guidelines, creativity in design and problem-solving will be highly valued.

Please ensure you read all task requirements carefully and refer to the provided resources to enhance your understanding and application of UI design concepts. Good luck!

**WEEK-03**

### Overview of HTML

HTML which stands for **H**yper**T**ext **M**arkup **L**anguage, is the language used to describe the structure of documents which are meant to be displayed on the web. To this effect, any content that you see on a website has some HTML code behind it.

Once you have written the code in any of these tools, you have to save it with the extension .html.

This means that if the file that you wrote the code in is called firstwebsite then you will save it as firstwebsite.html. This file can then be opened with any browser of your choice.

popular examples of browsers include:

* Google Chrome
* Mozilla Firefox
* Safari
* Microsoft Edge

Browsers are able to understand the HTML code and interpret it by rendering it to what you usually see when you visit a website. This means that each webpage is created by writing code which the browser converts to the actual content that you see on the page.

The HTML code itself basically consists of a series of elements and attributes which are used to mark up all of the components of a document to structure it in a meaningful way.

## **Elements and Attributes**

Now, let’s talk a bit more about elements and attributes, which are like the building blocks and stickers of our HTML document

In HTML, attributes help us give more information about our elements, like making a link open in a new tab or changing the size of our picture.

When we put these elements and attributes together in the right order, using our code editor, we start building our website, piece by piece.

It’s like how you would follow instructions to build a model airplane or a LEGO set. Each piece has its place, and when put together correctly, they create something amazing!

Lastly, it’s important to remember that the way we arrange our blocks – or in our case, elements – makes a big difference. Just like how a well-organized room is easier to move around in than a cluttered one, a well-structured website is easier for people to use and for computers to understand. That’s why learning HTML is a bit like learning the rules of a new game; it’s all about placing your elements and attributes in the right order to build something great.

## **Tags**

In our discussion about elements and attributes, there’s another important concept we need to understand: tags. Tags are like labels that we put on our blocks (elements) to tell the computer what they are supposed to do. Every tag has a name that reflects what the element does. For example, a tag for adding a paragraph of text is called a p tag.

Most tags come in pairs: an opening tag and a closing tag.

* The opening tag starts with < and ends with >, like this: <p> for a paragraph.
* The closing tag is almost the same, but it has a forward slash before the tag name, like this: </p>.

The text or content goes between these two tags. Think of it as putting a toy (your content) into a box (the element) and labeling both the top and bottom of the box with what’s inside (the opening and closing tags). This tells the computer where the element starts and ends.

For example, if I want to write “Hello, world!” on my webpage, I would use the paragraph tags like this:

<p>Hello, world!</p>

This tells the computer that “Hello, world!” is a paragraph.

## **How to use Elements and Attributes in your HTML code**

Now that we know about tags, let’s explore more about elements and attributes and how they appear in our HTML code. Remember, an element in HTML is like a block or a piece of a puzzle that makes up part of our webpage. Each element has a specific job, whether it’s to display text, link to another page, show an image, and so on.

### What is an Element?

An element is made up of several parts:

* the opening tag,
* the content, and
* the closing tag.

For example, a simple text element looks like this:

<p>This is a paragraph.</p>

Here, <p> is the opening tag that starts the element, This is a paragraph. is the content that the element displays, and </p> is the closing tag that ends the element.

The whole thing together, from the opening tag to the closing tag, is called an **element**.

### Where Do Attributes Go?

Attributes provide additional information about an element’s content, appearing within the opening tag. They are usually written in the form of name=value.

Attributes always have a name and a value, separated by an equals sign. The value is always enclosed in quotation marks. For example, inhref="https://example.com", href is the attribute name, and https://example.com is the attribute value. This tells the browser where the link goes.

They are like extra instructions that tell the browser more about how to display or handle the element. For example, if we want to make a link that opens in a new browser tab, we use an a element with an href attribute for the URL and a target attribute to specify the new tab:

<a href="https://example.com" target="\_blank">Visit Example.com</a>

|  |  |
| --- | --- |
| **Name of Attribute** | **Value** |
| href | https://example.com |
| target | \_blank |

In this case, <a> is the opening tag with two attributes inside it: href, which holds the link’s destination, and target, which tells the browser to open the link in a new tab (\_blank). The content, Visit Example.com, is what users click on, and </a> is the closing tag.

As you explore, remember that every website starts with simple tags and builds up from there. By the end of this week, you’ll be able to understand more and more of what you see when you inspect a website.

# Document Structure in HTML

Welcome to our journey through the structure of an HTML document! Think of an HTML document like a book. Just as a book has a cover, a table of contents, chapters, and pages, an HTML document has its own special parts that help organize everything inside it. Let’s discover what these parts are and how they work together to build a webpage.

## **The Basics of HTML Document Structure**

An HTML document is made up of several key parts: the <!DOCTYPE>, the <html> tag, the <head> section, and the <body> section. Each part has a special job, just like the different parts of a book.

### The <!DOCTYPE>

Our HTML document starts with something called the <!DOCTYPE>. This is not an element but a declaration that tells the web browser what version of HTML the page is written in. It’s like saying, “Hey browser, I’m going to speak in HTML5, so please understand me in that language.” For all our projects, we’ll use the HTML5 declaration, which looks like this:

<!DOCTYPE html>

### The <html> Tag

Next comes the <html> tag. This tag wraps around everything in our document, from start to finish. It’s like the cover of our book, holding everything inside. Inside the <html> tag, we divide our document into two main parts: the <head> and the <body>.

Inside the <html> tag, we tell the browser about the language we’re using with something called the lang attribute. This is like telling a friend which language you’ll be speaking so they can understand you better. For example, <html lang="en-US"> says, “This website is in American English.”

### The <head> Section

The <head> section is like the table of contents of our book. It contains information about the webpage, but it doesn’t have content that you see directly on the page. Instead, it has elements that tell the browser how to display the page or link to other files that the page needs. Here, we put the title of our webpage, link to stylesheets (which decide how our page looks), and include other information like the character set used by the page:

<head>

<title>My First Webpage</title>

<meta charset="UTF-8">

</head>

The content of the <title> tag appears in the browser tab and tells visitors and search engines what our page is about. It’s like the name of a book; it gives a hint about what’s inside.

### The <body> Section

Finally, we have the <body> section. This is where all the content that you see on the webpage lives. It’s like the chapters and pages of our book, filled with text, images, links, and more. Everything you want your users to see and interact with on your webpage goes here:

<body>

<h1>Welcome to My Webpage</h1>

<p>This is a paragraph on my webpage.</p>

<!-- More content like images, links, lists goes here -->

</body>

## **Putting It All Together**

When we put all these parts together, our HTML document looks something like this:

<!DOCTYPE html>

<html>

<head>

<title>My First Webpage</title>

<meta charset="UTF-8">

</head>

<body>

<h1>Welcome to My Webpage</h1>

<p>This is a paragraph on my webpage.</p>

<!-- More content goes here -->

</body>

</html>

This complete structure is the foundation of every webpage you visit. Just like every book is unique with its own table of contents and chapters, every webpage is unique with its own <head> and <body> content. But they all follow this basic structure to help organize and display information in a way that web browsers can understand.

### How to preview your HTML document

After creating your HTML document, it is now time to render the document and preview the output of the code.

There are two main ways to preview your HTML document:

**1. Using a Web Browser:**

This is the simplest and most common way. Here’s how to do it:

* **Save your HTML file:** Make sure your HTML document is saved with a .html extension.
* **Open the file in your browser:** Double-click the HTML file in your file explorer. This should automatically open it in your default web browser and render the page based on the HTML code.

**2. Using a Code Editor with Live Preview:**

Some code editors offer a live preview feature that lets you see the rendered HTML as you code. This can be a great way to see your changes reflected immediately without needing to constantly save and refresh in a browser. Here are a couple of options:

* **Built-in Preview:** Some code editors, like Visual Studio Code, have a built-in live preview feature. You can usually enable this feature in the editor’s settings.
* **Extensions:** Many code editors allow installing extensions that provide live preview functionality. These extensions may offer additional features like automatic refresh on save.

**Additional Tips:**

* **Keep your HTML and CSS files together:** If you’re using separate CSS files to style your HTML, make sure they are in the same folder or that the paths in your HTML code point to the correct location of the CSS files.

## **Practice Time**

Now that you know about the structure of an HTML document, try creating your own simple webpage. Use the structure we’ve learned to make a basic HTML document.

Include a title in the <head> section and some text in the <body> section. Preview the end product in a browser.

Top of Form

Bottom of Form

# Understanding Metadata in HTML

Welcome to the world of metadata in HTML! Think of metadata like the secret instructions you give to a web browser. These instructions help the browser understand how to show your website, who made it, and even what it’s about. Let’s dive into three important types of metadata: meta tags, character set, and viewport.

## **Meta Tags: The Secret Instructions**

Meta tags are like little notes you leave inside your website’s code. These notes tell the web browser important things it needs to know. You put these tags in the <head> section of your HTML document, which is like the brain of your website.

### What Can Meta Tags Tell Us?

* **Who created the website**: You can use a meta tag to say, “Hey, I made this!”
* **What your website is about**: This helps search engines understand your site better.
* **How to behave on different devices**: Some meta tags can tell the browser how to adjust your site so it looks great on phones, tablets, and computers.

## **Character Set: Speaking the Same Language**

The character set is a bit like choosing the alphabet for your website. The most common character set we use is UTF-8. This is a super versatile alphabet because it includes pretty much every character from every language in the world – yes, even emojis! 🎉

By using <meta charset="utf-8">, you’re telling the browser, “Use this universal alphabet so everyone can read my website, no matter what language they speak.”

## **Viewport: Making Sure Your Site Looks Good Everywhere**

The viewport is a special meta tag that makes sure your website looks good on any device. Have you ever visited a website on your phone and everything was super tiny? That’s because the website wasn’t using a viewport meta tag.

By adding <meta name="viewport" content="width=device-width, initial-scale=1.0"> to your website, you’re telling the browser, “Please adjust the size of my website so it fits perfectly on whatever screen it’s being viewed on.” This way, whether someone visits your site on a tiny phone or a big computer screen, it’ll look just right.

## **Practice Time!**

Now that we know about these secret instructions, let’s practice adding some to our own website. Try creating a simple HTML document and include meta tags for the character set and viewport. Don’t forget to give your site a title too! Here’s a quick start:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>My Awesome Site</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<!-- Add more meta tags here as you learn about them! -->

</head>

<body>

<h1>Welcome to My Awesome Site</h1>

<p>This is where my content will go.</p>

</body>

</html>

By understanding and using metadata effectively, you’re on your way to creating websites that are not just functional but also friendly to both browsers and visitors. Happy coding!

# Introduction to Semantic HTML

Imagine you’re organizing a big box of toys. Some toys are for playing catch, like balls, while others are for building, like blocks. Just like you organize your toys to find them easily, we use Semantic HTML to organize the content on our website so computers and people can understand it better.

## **What is Semantic HTML?**

Semantic HTML uses special tags that describe what type of content they contain, not just what they look like. For example, instead of using a generic box (a <div>tag) for everything, we use a specific box like a toy chest (a <nav> tag for navigation) or a bookshelf (an <article> tag for articles).

### Why is it Important?

1. **For People**: It helps users with disabilities to navigate and understand your website better. Imagine if someone couldn’t see the toys but could feel them. If the toys are well-organized, they can easily understand what each toy is for. Screen readers (tools that read out loud what’s on a computer screen) work similarly by reading web pages to users. Semantic HTML helps these tools understand the page better, making it easier for everyone to find what they need.
2. **For Computers**: Search engines, like Google, use robots to look at your website and decide what it’s about. Using the right boxes (tags) helps these robots understand your website better. This can make your website show up higher in search results when someone is looking for something you have. This is called Search Engine Optimization, or SEO.

## **Examples of Semantic HTML Tags**

* <header>: Like the top shelf where you keep your most important toys, this is for the top part of your website, usually with the title and navigation links.
* <nav>: This is like a map of your toy box, guiding you to different parts of your website.
* <article>: Each of these is like a book or a story, containing a complete piece of content that makes sense on its own.
* <footer>: The bottom of your toy box, where you might keep instructions or the box’s label. This is for information at the bottom of your website, like contact info or copyrights.

Semantic HTML helps us structure our web content in a similar, thoughtful way.

## **The <article> Tag**

Imagine writing a story or drawing a picture that stands on its own, even if it’s part of a bigger book or gallery. In HTML, we use the <article> tag for these kinds of stand-alone pieces. This could be a blog post, a news article, or even a comment.

### Example of <article>:

<article>

<h2>My Trip to the Moon</h2>

<p>Last summer, I dreamt I visited the moon. It was an incredible experience...</p>

</article>

## **The <section> Tag**

Now, think about chapters in a book or different areas in a playground. Each part has its own theme or purpose. The <section> tag helps us group related content together on our webpage, like a chapter or an area that serves a specific purpose.

### Example of <section>:

<section>

<h2>Favorite Space Movies</h2>

<p>Space has always fascinated filmmakers. Here are my top picks...</p>

</section>

## **The <aside> Tag**

Have you ever seen a textbox on the side of a page that gives you extra information? That’s what the <aside> tag is for. It’s like a sidebar or a callout that contains information only somewhat related to the main content, such as a biography of an author next to their article or related links.

### Example of <aside>:

<aside>

<h3>Did You Know?</h3>

<p>The moon moves away from Earth by 3.8 cm every year!</p>

</aside>

## **The <figure> Tag**

When you see a picture with a caption in a book or an article, that’s what <figure> and <figcaption> tags do in HTML. They work together to showcase illustrations, diagrams, photos, code listings, etc., with an accompanying explanation or caption.

### Example of <figure> and <figcaption>:

<figure>

<img src="moon-landing.jpg" alt="Astronauts landing on the moon">

<figcaption>Astronauts landing on the moon, captured in 1969.</figcaption>

</figure>

## **Practice Time!**

Let’s put these elements to work by creating a mini-blog page about an interesting topic. Use <article> for your blog posts, <section> to organize different categories or themes, <aside> for fun facts or related information, and <figure> to add images with captions.

Here’s a simple structure to get you started:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>My Mini-Blog on Space</title>

</head>

<body>

<header>

<h1>Space: The Final Frontier</h1>

</header>

<article>

<h2>Life on Mars?</h2>

<p>Scientists are exploring the possibility of life on Mars...</p>

<aside>

<h3>Mars Fact</h3>

<p>Mars has the largest dust storms in the solar system!</p>

</aside>

<figure>

<img src="mars.jpg" alt="The surface of Mars">

<figcaption>The red surface of Mars, as seen by the rover.</figcaption>

</figure>

</article>

<footer>

<p>Contact me at spacefan@example.com</p>

</footer>

</body>

</html>

By using Semantic HTML, you make your website more organized and accessible, helping everyone understand your content better, just like a well-organized toy box!

# Headings and Sections

Understanding the proper use of headings and sectioning elements is crucial for creating well-structured web content. These elements help both users and search engines navigate and understand the hierarchy and organization of your content. Let’s explore how to use headings and sections effectively, and how they impact SEO (Search Engine Optimization).

## **Headings: The Backbone of Content Structure**

Headings, from <h1> to <h6>, are like the titles and subtitles in a book, outlining the main topics and subtopics. They help readers grasp the main points of your content at a glance.

### Rules of Thumb for Headings:

1. **Use <h1> for Your Main Title**: Every page should have a main title, and it should be wrapped in an <h1> tag. This tells everyone, “Here’s the most important thing about this page.”
2. **Organize Headings in a Hierarchical Order**: Think of headings as a pyramid, with <h1> at the top. As you move down the page, use <h2> for main sections, <h3> for subsections under those, and so on. This hierarchy helps readers and search engines understand how your content is organized.
3. **Keep It Relevant and Clear**: Headings should give a clear idea of what the following section is about. This helps users find the information they’re looking for and helps search engines understand your content’s context and relevance.

## **Sectioning Elements: Organizing Your Content**

Sectioning elements, such as <section>, <article>, <nav>, and <aside>, provide a way to group related content. They’re like chapters in a book, each dedicated to a specific topic or theme.

### How to Use Sectioning Elements:

* **<section>**: Use this for thematic groupings of content. If a part of your content stands out as a separate topic, wrap it in a <section> tag.
* **<article>**: This is for self-contained content that could theoretically stand alone, like a blog post or a news article.
* **<nav>**: Dedicated to navigation links. This helps users and search engines find their way around your site.
* **<aside>**: For content that’s related but not part of the main content flow, like sidebars, related links, or author bios.

## **The Impact on SEO**

Proper use of headings and sectioning elements significantly impacts SEO. Here’s how:

* **Improves Page Structure**: Search engines use headings to index the structure and content of your web pages. Well-organized content is more likely to rank higher in search results.
* **Enhances Accessibility**: Screen readers use headings to navigate content, improving accessibility for users with visual impairments. Accessible sites are favored by search engines.
* **Facilitates Content Discovery**: Clear headings and well-structured content make it easier for search engines to understand and accurately index your content, making it more discoverable to users.

## **Practice Time!**

Create a webpage about a hobby of yours, using headings and sectioning elements to organize the content. Here’s a simple example to get you started:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>My Love for Astronomy</title>

</head>

<body>

<h1>Astronomy: The Gateway to the Stars</h1>

<nav>

<h2>Navigation</h2>

<ul>

<li><a href="#why-astronomy">Why I Love Astronomy</a></li>

<li><a href="#getting-started">Getting Started</a></li>

</ul>

</nav>

<section>

<h2>Why I Love Astronomy</h2>

<p>Astronomy allows us to explore the vastness of the universe from our backyards...</p>

</section>

<section>

<h2>Getting Started with Astronomy</h2>

<p>Starting with astronomy is easier than you might think. Here are some tips to get you started...</p>

</section>

<footer>

<h2>Contact</h2>

<p>If you're interested in astronomy and want to chat, email me at starwatcher@example.com!</p>

</footer>

</body>

</html>

By carefully structuring your content with headings and sections, you make your website more navigable and understandable, both for your users and for search engines. This thoughtful organization is key to creating effective, accessible, and SEO-friendly web content.

# Attributes & Types of Attributes

Attributes in HTML are like the special features or settings you can apply to elements to define their behavior, appearance, or provide additional information. Let’s dive into some important types of attributes you’ll use in your HTML documents: global attributes, id, class, style, and custom data attributes.

## **Global Attributes**

Global attributes are like universal remote controls; they can work with almost any HTML element. These attributes provide essential functionalities or information applicable across various elements.

### Examples of Global Attributes:

* **title**: Offers additional information about an element, typically displayed as a tooltip when the mouse hovers over the element.
* **tabindex**: Specifies the order in which elements will receive focus when navigating through them using the keyboard. This is crucial for accessibility.
* **lang**: Declares the language of the element’s content, important for translation tools and screen readers.
* **hidden**: Indicates that the element is not presently visible on the page.

## **The id Attribute**

The id attribute assigns a unique identifier to an element. Think of it as giving a specific name to an element so you can easily find it in a crowd. This is especially useful when you want to link to a specific part of a webpage or manipulate an element with CSS or JavaScript.

### Example of id:

<p id="introduction">This is the introduction to our topic.</p>

## **The class Attribute**

The class attribute groups multiple elements under the same identifier, allowing you to style or manipulate them as a collective. Imagine assigning a team jersey to players so you can easily identify them as part of your team.

### Example of class:

<p class="highlight">This text will stand out because of its class.</p>

## **The style Attribute**

The style attribute lets you apply CSS rules directly to an element. It’s like customizing the look of your element directly on the spot without needing a separate stylesheet.

### Example of style:

<p style="color: blue;">This text will be blue.</p>

## **Custom Data Attributes**

Custom data attributes (data-\*) allow you to store extra information with an element without any visual representation. These attributes are perfect for keeping track of data that JavaScript might need to enhance the user experience.

### Example of Custom Data Attributes:

<div data-product-id="12345" data-price="25">This is a product.</div>

## **Practice Time!**

Now that we’ve explored these attributes, let’s practice by creating a simple user profile card. We’ll use id and class for styling, style to apply direct styles, and custom data attributes to hold additional information.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>User Profile</title>

<style>

.user-card {

background-color: #f0f0f0;

padding: 20px;

margin: 10px 0;

}

.highlight {

color: green;

}

</style>

</head>

<body>

<div id="user123" class="user-card" data-user-level="novice" style="border: 1px solid #000;">

<p class="highlight">Name: Jane Doe</p>

<p>Interests: HTML, CSS, JavaScript</p>

<p>Email: jane.doe@example.com</p>

</div>

</body>

</html>

In this exercise, we’ve created a user profile card that utilizes id for unique identification, class for styling groups of elements, inline style for specific styling, and custom data-\* attributes to store extra information about the user. This practice helps in understanding how different attributes enhance the functionality and presentation of HTML elements.

# Text basics

Exploring the basics of text in HTML is like learning the alphabet before you can start writing stories. It’s essential for creating content that is not only visually appealing but also structurally sound and accessible. Let’s dive into paragraphs, formatting text, and how to use special characters in your web pages.

## **Paragraphs: The Foundation of Web Content**

In HTML, paragraphs are marked with the <p> tag. Think of paragraphs as the building blocks of your web content, organizing your text into readable chunks. Each <p> tag creates a new paragraph, and browsers automatically add some space before and after paragraphs to help separate them visually.

### Example of a Paragraph:

<p>This is a paragraph of text in HTML. It introduces a single idea or point.</p>

## **Formatting Text: Adding Emphasis and Importance**

When you’re writing, sometimes you want to make certain words stand out to convey emphasis or importance. HTML provides several tags for formatting text, each with its own meaning and use case.

### Bold and Italic

* **Bold (<strong>):** Use the <strong> tag to indicate that text is important or should be paid special attention to. This is often displayed as bold text.

<p>I <strong>strongly</strong> recommend practicing HTML daily.</p>

* **Italic (<em>):** The <em> tag is used to emphasize text, which is typically displayed as italicized. It can change the meaning of a sentence or indicate a different tone.

<p>To <em>really</em> understand HTML, you need to build projects.</p>

### Subscript and Superscript

* **Subscript (<sub>):** Used for characters that should appear half a letter below the normal line, like chemical formulas or mathematical notation.

<p>Water is H<sub>2</sub>O.</p>

* **Superscript (<sup>):** Used for characters that should be half a letter above the normal line, such as mathematical exponents.

<p>E=mc<sup>2</sup>.</p>

## **Special Characters: Adding Flair and Function**

Sometimes, you need to include characters in your HTML that are reserved for specific functions in HTML syntax, or you want to display symbols, currency signs, or letters from different languages. These are called “special characters” or “HTML entities.”

### Examples of Special Characters:

* **Ampersand (&amp;):** Since the ampersand symbol (&) is used in HTML to denote the beginning of an entity, to display it, you use &amp;.

<p>Ben &amp; Jerry's</p>

* **Less-than and greater-than (&lt; and &gt;):** These symbols are used in HTML to create tags, so to display them, you use &lt; for < and &gt; for >.

<p>To add a paragraph, use the &lt;p&gt; tag.</p>

* **Non-breaking Space (&nbsp;):** Sometimes, you want to add extra spaces that the browser won’t collapse into a single space. Use &nbsp; for each extra space you need.

<p>This sentence uses&nbsp;&nbsp;&nbsp;multiple spaces.</p>

* **Copyright (&copy;), Trademark (&trade;), and Registered (&reg;):** To include these symbols, use their corresponding entities.

<p>&copy; 2024 My Website. All rights reserved.</p>

## **Practice Time!**

Let’s put these concepts into practice by creating a simple HTML document that uses paragraphs for text, applies formatting for emphasis, and includes special characters.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Text Basics in HTML</title>

</head>

<body>

<p>This is an example of a paragraph in HTML. Notice how it makes the text easier to read.</p>

<p>I <strong>strongly</strong> recommend using <em>emphasis</em> tags like <strong>strong</strong> and <em>em</em> to highlight important text. Also, don't forget about <sub>subscript</sub> and <sup>superscript</sup> for details like H<sub>2</sub>O and E=mc<sup>2</sup>.</p>

<p>Special characters like &amp;, &lt;, &gt;, and &nbsp; are essential for web development. Remember to use &copy; or &reg; for copyright or registered symbols, respectively.</p>

</body>

</html>

By mastering these text basics, you’ll be well on your way to creating rich, accessible, and engaging web content.

# Links in HTML

Links, or hyperlinks, are the essence of the web, connecting pieces of content across the internet. Understanding how to create and use links with absolute and relative URLs, anchor tags, and the target attribute is crucial for web development. Let’s explore these concepts to ensure you can effectively link content within your web projects.

## **Absolute vs Relative URLs**

When you create a link, you can use either an absolute URL or a relative URL to specify the link’s destination.

### Absolute URLs

An absolute URL provides the complete address of a webpage, including the protocol (http:// or https://), domain name, and path to the resource. It’s like giving someone your full address with the country, city, street, and house number.

#### Example of an Absolute URL:

<a href="https://www.example.com/about.html">About Us</a>

This link will always lead to the “About Us” page on www.example.com, no matter where it’s used.

### Relative URLs

A relative URL specifies the path to a resource in relation to the current document’s location. It’s akin to directing someone to the kitchen from the living room within the same house.

#### Example of a Relative URL:

<a href="/about.html">About Us</a>

If your current URL is https://www.example.com/products/widget.html, this link will take you to https://www.example.com/about.html. Relative URLs are useful for linking to content within the same website because they make maintenance easier; if the domain name changes, the links will still work.

## **Anchor Tags**

The anchor tag (<a>) is used to create links in HTML. It has an href attribute (hypertext reference) that specifies the URL of the page the link goes to.

### Example of an Anchor Tag:

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

## **The target Attribute**

The target attribute of an anchor tag specifies where to open the linked document. It’s useful for controlling the user experience, such as opening a link in a new tab.

### Values for the target Attribute:

* **\_self**: Opens the link in the current tab (this is the default behavior).
* **\_blank**: Opens the link in a new tab or window. It’s particularly useful for external links, ensuring users don’t lose their place on your site.
* **\_parent**: Opens the link in the parent frame (used in framed sites, which are less common now).
* **\_top**: Opens the link in the full body of the window, clearing all frames (also used in framed sites).

#### Example of Using the target Attribute:

<a href="https://www.example.com" target="\_blank">Visit Example in a New Tab</a>

## **Practice Time!**

Let’s practice creating links with different types of URLs and using the targetattribute. Imagine you’re building a personal blog and want to include a link to your GitHub profile in a new tab and a link to your contact page located on the same site.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>My Personal Blog</title>

</head>

<body>

<h1>Welcome to My Blog</h1>

<p>Check out my <a href="https://github.com/yourusername" target="\_blank">GitHub profile</a> for more projects.</p>

<p>Feel free to reach out via the <a href="/contact.html">contact page</a> if you have any questions.</p>

</body>

</html>

By understanding and utilizing absolute and relative URLs, anchor tags, and the target attribute, you can create intuitive and user-friendly navigation within and outside your website.

# Lists

Lists are fundamental components in HTML that allow you to organize information clearly and logically. There are three main types of lists in HTML: ordered lists, unordered lists, and description lists. Each serves a different purpose depending on how you want to present your information.

## **Ordered Lists**

Ordered lists (<ol>) are used when the sequence of the items is important. It’s like when you’re listing the steps in a recipe or the top ten movies of the year. The list items are marked with numbers by default.

### Example of an Ordered List:

<ol>

<li>Preheat the oven to 350°F (175°C).</li>

<li>Mix flour, sugar, and eggs in a bowl.</li>

<li>Pour the mixture into a baking pan.</li>

<li>Bake for 30 minutes.</li>

</ol>

## **Unordered Lists**

Unordered lists (<ul>) are used when the order of the items doesn’t matter. This is similar to listing your favorite foods or things to pack for a vacation. The list items are typically marked with bullets.

### Example of an Unordered List:

<ul>

<li>Pizza</li>

<li>Ice cream</li>

<li>Chocolate</li>

</ul>

## **Description Lists**

Description lists (<dl>) are a bit different. They are used to pair terms with their descriptions, similar to a glossary or a dictionary. This structure involves three tags: <dl> for the list, <dt> for the term, and <dd> for the description.

### Example of a Description List:

<dl>

<dt>HTML</dt>

<dd>Hypertext Markup Language - the standard markup language for creating web pages.</dd>

<dt>CSS</dt>

<dd>Cascading Style Sheets - a stylesheet language used to describe the presentation of a document written in HTML.</dd>

</dl>

## **Practice Time!**

Let’s put these list types into practice by creating a webpage that includes an ordered list for a simple recipe, an unordered list for kitchen tools needed, and a description list for cooking terms used in the recipe.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Simple Cookie Recipe</title>

</head>

<body>

<h1>Simple Cookie Recipe</h1>

<h2>Ingredients</h2>

<ul>

<li>Flour</li>

<li>Sugar</li>

<li>Butter</li>

</ul>

<h2>Directions</h2>

<ol>

<li>Preheat your oven to 375°F (190°C).</li>

<li>Mix all ingredients in a large bowl.</li>

<li>Place spoonfuls of dough on a baking sheet.</li>

<li>Bake for 10 to 12 minutes.</li>

</ol>

<h2>Cooking Terms</h2>

<dl>

<dt>Preheat</dt>

<dd>Heat (an oven or grill) beforehand.</dd>

<dt>Bake</dt>

<dd>Cook (food) by dry heat without direct exposure to a flame, typically in an oven.</dd>

</dl>

</body>

</html>

By incorporating ordered, unordered, and description lists in your HTML documents, you can effectively organize and present a wide range of information in a structured and accessible manner.

# Lists

Lists are fundamental components in HTML that allow you to organize information clearly and logically. There are three main types of lists in HTML: ordered lists, unordered lists, and description lists. Each serves a different purpose depending on how you want to present your information.

## **Ordered Lists**

Ordered lists (<ol>) are used when the sequence of the items is important. It’s like when you’re listing the steps in a recipe or the top ten movies of the year. The list items are marked with numbers by default.

### Example of an Ordered List:

<ol>

<li>Preheat the oven to 350°F (175°C).</li>

<li>Mix flour, sugar, and eggs in a bowl.</li>

<li>Pour the mixture into a baking pan.</li>

<li>Bake for 30 minutes.</li>

</ol>

## **Unordered Lists**

Unordered lists (<ul>) are used when the order of the items doesn’t matter. This is similar to listing your favorite foods or things to pack for a vacation. The list items are typically marked with bullets.

### Example of an Unordered List:

<ul>

<li>Pizza</li>

<li>Ice cream</li>

<li>Chocolate</li>

</ul>

## **Description Lists**

Description lists (<dl>) are a bit different. They are used to pair terms with their descriptions, similar to a glossary or a dictionary. This structure involves three tags: <dl> for the list, <dt> for the term, and <dd> for the description.

### Example of a Description List:

<dl>

<dt>HTML</dt>

<dd>Hypertext Markup Language - the standard markup language for creating web pages.</dd>

<dt>CSS</dt>

<dd>Cascading Style Sheets - a stylesheet language used to describe the presentation of a document written in HTML.</dd>

</dl>

## **Practice Time!**

Let’s put these list types into practice by creating a webpage that includes an ordered list for a simple recipe, an unordered list for kitchen tools needed, and a description list for cooking terms used in the recipe.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Simple Cookie Recipe</title>

</head>

<body>

<h1>Simple Cookie Recipe</h1>

<h2>Ingredients</h2>

<ul>

<li>Flour</li>

<li>Sugar</li>

<li>Butter</li>

</ul>

<h2>Directions</h2>

<ol>

<li>Preheat your oven to 375°F (190°C).</li>

<li>Mix all ingredients in a large bowl.</li>

<li>Place spoonfuls of dough on a baking sheet.</li>

<li>Bake for 10 to 12 minutes.</li>

</ol>

<h2>Cooking Terms</h2>

<dl>

<dt>Preheat</dt>

<dd>Heat (an oven or grill) beforehand.</dd>

<dt>Bake</dt>

<dd>Cook (food) by dry heat without direct exposure to a flame, typically in an oven.</dd>

</dl>

</body>

</html>

By incorporating ordered, unordered, and description lists in your HTML documents, you can effectively organize and present a wide range of information in a structured and accessible manner.

# Lists

Lists are fundamental components in HTML that allow you to organize information clearly and logically. There are three main types of lists in HTML: ordered lists, unordered lists, and description lists. Each serves a different purpose depending on how you want to present your information.

## **Ordered Lists**

Ordered lists (<ol>) are used when the sequence of the items is important. It’s like when you’re listing the steps in a recipe or the top ten movies of the year. The list items are marked with numbers by default.

### Example of an Ordered List:

<ol>

<li>Preheat the oven to 350°F (175°C).</li>

<li>Mix flour, sugar, and eggs in a bowl.</li>

<li>Pour the mixture into a baking pan.</li>

<li>Bake for 30 minutes.</li>

</ol>

## **Unordered Lists**

Unordered lists (<ul>) are used when the order of the items doesn’t matter. This is similar to listing your favorite foods or things to pack for a vacation. The list items are typically marked with bullets.

### Example of an Unordered List:

<ul>

<li>Pizza</li>

<li>Ice cream</li>

<li>Chocolate</li>

</ul>

## **Description Lists**

Description lists (<dl>) are a bit different. They are used to pair terms with their descriptions, similar to a glossary or a dictionary. This structure involves three tags: <dl> for the list, <dt> for the term, and <dd> for the description.

### Example of a Description List:

<dl>

<dt>HTML</dt>

<dd>Hypertext Markup Language - the standard markup language for creating web pages.</dd>

<dt>CSS</dt>

<dd>Cascading Style Sheets - a stylesheet language used to describe the presentation of a document written in HTML.</dd>

</dl>

## **Practice Time!**

Let’s put these list types into practice by creating a webpage that includes an ordered list for a simple recipe, an unordered list for kitchen tools needed, and a description list for cooking terms used in the recipe.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Simple Cookie Recipe</title>

</head>

<body>

<h1>Simple Cookie Recipe</h1>

<h2>Ingredients</h2>

<ul>

<li>Flour</li>

<li>Sugar</li>

<li>Butter</li>

</ul>

<h2>Directions</h2>

<ol>

<li>Preheat your oven to 375°F (190°C).</li>

<li>Mix all ingredients in a large bowl.</li>

<li>Place spoonfuls of dough on a baking sheet.</li>

<li>Bake for 10 to 12 minutes.</li>

</ol>

<h2>Cooking Terms</h2>

<dl>

<dt>Preheat</dt>

<dd>Heat (an oven or grill) beforehand.</dd>

<dt>Bake</dt>

<dd>Cook (food) by dry heat without direct exposure to a flame, typically in an oven.</dd>

</dl>

</body>

</html>

By incorporating ordered, unordered, and description lists in your HTML documents, you can effectively organize and present a wide range of information in a structured and accessible manner.

# Navigation In HTML

Creating a navigable page layout is crucial for enhancing user experience on a website. It allows users to easily find the information they’re looking for and understand the structure of your site. The <nav> element in HTML plays a key role in defining navigation sections, typically containing a list of links to other pages or sections within the same page.

## **The <nav> Element**

The <nav> element is specifically meant to wrap major navigation blocks in your HTML document. While you can have multiple <nav> elements on a page (for example, one for the main site navigation and another for a sidebar or footer links), it’s primarily used for the main navigation menu.

### Best Practices for Using <nav>:

1. **Contain Links**: The <nav> element should primarily contain hyperlinks to other pages or sections within the page.
2. **Not for Every Link**: Not all groups of links on a page need to be inside a <nav> element. Use it for major navigation areas. Other links can simply be grouped using a <div> or other elements.
3. **Accessibility**: Using <nav> helps screen readers and other assistive technologies understand the structure of your website, making it more accessible.

## **Creating a Basic Navigation Bar**

A basic navigation bar typically includes a list of links styled with CSS. Here’s a simple example:

### HTML

<nav>

<ul>

<li><a href="index.html">Home</a></li>

<li><a href="about.html">About Us</a></li>

<li><a href="services.html">Services</a></li>

<li><a href="contact.html">Contact</a></li>

</ul>

</nav>

### CSS (for basic styling)

nav ul {

list-style-type: none;

margin: 0;

padding: 0;

overflow: hidden;

background-color: #333;

}

nav li {

float: left;

}

nav li a {

display: block;

color: white;

text-align: center;

padding: 14px 16px;

text-decoration: none;

}

nav li a:hover {

background-color: #111;

}

## **Responsive Navigation**

Responsive navigation adapts to different screen sizes, ensuring a good user experience across devices. A common approach is to use a “hamburger menu” on smaller screens, which expands to show navigation links when clicked. Implementing this requires more advanced HTML, CSS, and often JavaScript.

## **Practice Time!**

Let’s create a simple webpage with a basic navigation bar that links to hypothetical pages of a website.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>My Website</title>

<style>

/\* Add the CSS styling here \*/

nav ul {

list-style-type: none;

margin: 0;

padding: 0;

overflow: hidden;

background-color: #333;

}

nav li {

float: left;

}

nav li a {

display: block;

color: white;

text-align: center;

padding: 14px 16px;

text-decoration: none;

}

nav li a:hover {

background-color: #111;

}

</style>

</head>

<body>

<nav>

<ul>

<li><a href="index.html">Home</a></li>

<li><a href="about.html">About Us</a></li>

<li><a href="services.html">Services</a></li>

<li><a href="contact.html">Contact</a></li>

</ul>

</nav>

<h1>Welcome to My Website</h1>

<p>This is a demonstration of a simple navigable page layout using the nav element.</p>

</body>

</html>

By incorporating the <nav> element and organizing your links effectively, you create a navigable structure that improves the overall usability and accessibility of your website.

# Tables in HTML

Tables in HTML are a way to organize data in rows and columns, similar to how you might see information arranged in a spreadsheet. Understanding how to create tables, use headers, and format tables for accessibility is crucial for presenting complex information on web pages. Let’s dive into the syntax, structure, and best practices for creating accessible tables.

## **Basic Table Syntax**

A table is created with the <table> element.

Inside, you use

 <tr> (table row),

<td> (table cell), and

 <th> (table header) elements to define the structure.

### Example of a Simple Table:

<table>

<tr>

<th>Month</th>

<th>Savings</th>

</tr>

<tr>

<td>January</td>

<td>$100</td>

</tr>

<tr>

<td>February</td>

<td>$80</td>

</tr>

</table>

## **Headers**

The <th> element is used to define header cells in tables, which contain explanatory text for the column or row. Browsers typically display header cells in bold and centered.

### Specifying Scope:

For accessibility, it’s important to specify whether a header cell is for a row, a column, or a group of rows or columns using the scope attribute. This helps screen readers understand the relationship between header cells and data cells.

* **Column header**: scope="col" (default for <th> elements)
* **Row header**: scope="row"

### Example with Scope:

<table>

<tr>

<th scope="col">Month</th>

<th scope="col">Savings</th>

</tr>

<tr>

<th scope="row">January</th>

<td>$100</td>

</tr>

</table>

## **Merging Cells**

You can merge cells across rows or columns to group information together using the rowspan and colspan attributes.

* **rowspan**: Merges cells vertically across rows.
* **colspan**: Merges cells horizontally across columns.

### Example of Merging Cells:

<table>

<tr>

<th>Person</th>

<th colspan="2">Details</th>

</tr>

<tr>

<td>John Doe</td>

<td>Age: 30</td>

<td>Occupation: Developer</td>

</tr>

</table>

## **Table Accessibility**

Accessibility in tables is about ensuring that all users, including those using assistive technologies like screen readers, can understand the content and structure of your tables.

### Tips for Accessible Tables:

* **Use <th> elements with the scope attribute** to define headers for rows and columns.
* **Provide captions** with the <caption> element to offer a summary or context for the table.
* **Avoid complex nested tables** if possible, as they can be challenging to navigate with a screen reader.
* **Use semantic markup correctly** to ensure the content is understandable in the absence of visual cues.

### Example with Caption and Accessibility Features:

<table>

<caption>Monthly Savings</caption>

<tr>

<th scope="col">Month</th>

<th scope="col">Savings</th>

</tr>

<tr>

<th scope="row">January</th>

<td>$100</td>

</tr>

<tr>

<th scope="row">February</th>

<td>$80</td>

</tr>

</table>

## **Practice Task**

Create a table that lists three of your favorite books, including columns for the book title, author, year published, and genre. Use header cells for column headings, include a caption for your table, and merge cells where appropriate to demonstrate your understanding of table syntax and accessibility practices.

Understanding and applying these concepts will enable you to create tables that are not only visually appealing but also accessible to all users, enhancing the overall user experience on your web pages.

# Forms in HTML

Forms are a critical component of web development, allowing users to submit data to a web server for processing. Understanding form structure, various input types, labels, placeholders, buttons, and validation is essential for creating effective and user-friendly forms. This guide will cover these topics, providing a solid foundation for building functional web forms.

## **Form Structure**

A form is defined with the <form> element. It can contain various form elements like inputs, labels, buttons, etc. The action attribute specifies where to send the form data when submitted, and the method attribute defines the HTTP method (usually “get” or “post”).

### Basic Form Example:

<form action="/submit-form" method="post">

<!-- Form elements go here -->

</form>

## **Input Types**

HTML offers a variety of <input> types for different data needs, including:

* text: For basic text input.
* password: For password input that should be obscured.
* email: For email addresses, with basic validation.
* number: For numerical input.
* date: For selecting dates.
* checkbox: For boolean choices.
* radio: For multiple choices where only one option is selectable.
* submit: For submitting the form.

### Example of Different Input Types:

<input type="text" name="username" placeholder="Username">

<input type="password" name="password" placeholder="Password">

<input type="email" name="email" placeholder="Email">

## **Labels and Placeholders**

Labels (<label>) are used to identify form elements, improving accessibility and usability. The for attribute of a label should match the id of the corresponding input.

Placeholders provide a hint or example of what the user should enter, displayed within the input field until the user begins typing.

### Example with Labels and Placeholders:

<label for="username">Username:</label>

<input type="text" id="username" name="username" placeholder="Enter your username">

<label for="email">Email:</label>

<input type="email" id="email" name="email" placeholder="example@domain.com">

## **Buttons**

Buttons are used within forms to submit data or perform an action. The <button>element or <input type="submit"> can be used to create a submit button.

### Example of a Submit Button:

<button type="submit">Submit</button>

<!-- or -->

<input type="submit" value="Submit">

## **Form Validation**

Validation ensures that users fill out forms in the correct format. HTML5 introduces built-in validation for form elements, which can be enhanced with custom JavaScript validation for more complex scenarios.

### Basic HTML Validation:

* required: Specifies that an input field must be filled out.
* minlength and maxlength: Define the minimum and maximum length for text inputs.
* pattern: Uses a regular expression to define a pattern that the input must match.

### Example with Validation:

<form action="/submit-form" method="post">

<label for="username">Username (4-8 characters):</label>

<input type="text" id="username" name="username" required minlength="4" maxlength="8" placeholder="Username">

<label for="email">Email:</label>

<input type="email" id="email" name="email" required placeholder="example@domain.com">

<button type="submit">Submit</button>

</form>

## **Practice Task**

Create a registration form for an online service that includes the following elements:

1. **Username**: Text input, required, with a suitable length limit.
2. **Password**: Password input, required, with minimum length validation.
3. **Email**: Email input, required, with built-in email validation.
4. **Age**: Number input, optional, with a minimum and maximum value.
5. **Terms of Service**: Checkbox, required (user must check to proceed).
6. **Submit button**.

Ensure each input has an associated label and utilize placeholders to guide user input. Test your form to see how the browser handles validation for different fields.

By completing this task, you’ll gain practical experience in form creation, understanding input types, implementing labels and placeholders, using buttons, and applying basic validation techniques to improve user interaction and data integrity.

# Images in HTML

Images are a crucial part of web content, enhancing the visual appeal and supporting the textual content. Understanding how to add images correctly, use alternative text (alt attributes) for accessibility, and incorporate <figure> and <figcaption> elements for semantic grouping and captioning is essential for web developers.

## **Adding Images**

The <img> tag is used to embed an image in an HTML page. The src attribute specifies the path to the image file, while the alt attribute provides alternative text for the image, which is crucial for accessibility and SEO.

### Example of Adding an Image:

<img src="path/to/image.jpg" alt="Description of the image">

## **Alt Attributes**

The alt attribute is a brief description of the image, which screen readers read aloud to users with visual impairments. It’s also displayed if the image fails to load and helps search engines understand the content of the image.

### Best Practices for Alt Text:

* Be descriptive and concise.
* Avoid starting with “image of…” or “picture of…”; it’s already implied.
* Include functional information if the image is a link or a button.

### Example with Alt Attribute:

<img src="logo.png" alt="Company Logo">

## **The <figure> and <figcaption>Elements**

The <figure> element is used to semantically group media content with its caption or description. <figcaption> provides a caption or description for the content within <figure>. This grouping is particularly useful for images, diagrams, code snippets, etc., that are referenced in the main content.

### Example of Figure with Figcaption:

<figure>

<img src="path/to/chart.png" alt="Sales Chart for 2024">

<figcaption>Figure 1: Sales chart showing monthly sales for 2024.</figcaption>

</figure>

## **Practice Task**

Create an HTML document that includes:

1. **An Image**: Add an image relevant to a topic you’re interested in (e.g., a hobby). Ensure the image has an appropriate alt attribute.
2. **Figure and Figcaption**: Use the <figure> element to contain the image and provide a caption using <figcaption>. The caption should describe the context or significance of the image in relation to the topic.

Ensure the path to the image is correct, and the alt attribute accurately describes the image’s content for users who might not be able to see it.

By completing this task, you’ll practice incorporating visual elements into web pages in a way that enhances accessibility and provides context, making your web content more engaging and informative.

# Audio and Video

Embedding audio and video in web pages enriches the content and improves user engagement. Modern HTML5 makes this straightforward with the <audio> and <video> elements, providing built-in controls for playback and including features for accessibility. Let’s explore how to embed these media types, use the <source> element for multiple formats, and ensure accessibility.

## **Embedding Audio**

The <audio> element allows you to include sound content such as music, voice recordings, or podcasts directly into your web pages. To control playback, you can add the controls attribute, which displays the browser’s default play, pause, and volume controls.

### Basic Audio Embed:

<audio controls>

<source src="path/to/audio.mp3" type="audio/mpeg">

Your browser does not support the audio element.

</audio>

## **Embedding Video**

Similar to audio, the <video> element embeds video content. The controlsattribute adds playback controls. It’s good practice to include width and height attributes to specify the video’s size, though these can also be controlled with CSS.

### Basic Video Embed:

<video controls width="250">

<source src="path/to/video.mp4" type="video/mp4">

Your browser does not support the video tag.

</video>

## **Using the <source> Element**

The <source> element allows you to specify multiple media sources for the same audio or video, enabling the browser to choose a format it supports. This is useful for compatibility across different browsers.

### Example with Multiple Sources:

<video controls>

<source src="movie.mp4" type="video/mp4">

<source src="movie.ogg" type="video/ogg">

Your browser does not support the video tag.

</video>

## **Accessibility Features**

Ensuring that audio and video content is accessible to all users, including those with disabilities, is crucial. Here are some tips:

* **Transcripts for Audio**: Provide a text transcript for audio content, making it accessible to those who are deaf or hard of hearing.
* **Subtitles and Captions for Video**: Include subtitles or captions for video content for users who cannot hear the audio. This can be done with the <track> element inside the <video> tag.
* **Descriptive Audio**: Offer descriptive audio tracks for users who are visually impaired to describe visual elements of the video.

### Example with Captions:

<video controls>

<source src="movie.mp4" type="video/mp4">

<track src="captions\_en.vtt" kind="captions" srclang="en" label="English">

Your browser does not support the video tag.

</video>

## **Practice Task**

Create a web page that includes both an audio clip and a video clip related to a topic of your choice. Ensure to:

1. **Provide Multiple Sources** for each media type to ensure broad compatibility.
2. **Include Playback Controls** so users can easily play, pause, and adjust volume.
3. **Add Accessibility Features**:
   * For audio, include a transcript of the content.
   * For video, include captions (you can create a simple .vtt file for captions as an example).

This task will help you understand the importance of embedding media in web development and ensuring content is accessible and enjoyable for all users.

# Introduction to HTML

* Novice
* Weight: 1
* Project will start Sep 9, 2024 12:00 AM, must end by Sep 16, 2024 12:00 AM
* Checker was released at Sep 9, 2024 12:00 AM
* An auto review will be launched at the deadline

## **Learning Objectives**

At the end of this project, you are expected to be able to [explain to anyone](https://intranet.alxswe.com/rltoken/wMEI3Uq9YrfXo98kxNWU0w), **without the help of Google**:

### General

* What is HTML
* How to create an HTML page
* What is a markup language
* What is the DOM
* What is an element / tag
* What is an attribute
* How does the browser load a webpage

## **Requirements**

### General

* Recommended editors: Visual studio code
* All your files should end with a new line
* A README.md file, at the root of the folder of the project is mandatory
* You are **not allowed** to install, import or use external libraries. This website must be build with only HTML/CSS/JavaScript. No NodeJS, React, VueJS, Bootstrap, etc.
* Your code should be W3C compliant and validate with [W3C-Validator](https://intranet.alxswe.com/rltoken/k_qWagKGYKVC6FWS8E_vJQ)

### Quiz questions

#### Question #0

What is the primary purpose of the <meta> tags in HTML?

* To provide instructions to the web browser about the webpage, such as creator information and viewport settings
* To define the structure and layout of a webpage
* To add images and multimedia content to the webpage
* To create interactive elements for user engagement

#### Question #1

What is the purpose of the <aside> tag in Semantic HTML?

* To represent a standalone piece of content, such as a blog post or news article.
* To group related content together based on a common theme or purpose.
* To display extra information or related content somewhat separate from the main content.
* To showcase illustrations, diagrams, photos, or code listings with accompanying explanations or captions.

#### Question #2

How does Semantic HTML benefit people with disabilities?

* By providing visually appealing designs that enhance user experience.
* By using specific tags that assist screen readers in understanding and navigating website content.
* By optimizing websites for faster loading times and improved performance.
* By incorporating interactive elements that engage users with disabilities.

#### Question #3

What does the following meta tag indicate: <meta charset="utf-8">?

* It defines the color scheme of the webpage.
* It specifies the website’s language
* It selects the alphabet for encoding text characters on the webpage.
* It enhances the website’s compatibility with different browsers.

#### Question #4

Which meta tag ensures that a website looks good on any device by adjusting its size accordingly?

* <meta name="screen-resolution" content="auto">
* <meta name="device-size" content="adjust">
* <meta name="viewport" content="width=device-width, initial-scale=1.0">
* <meta name="responsive" content="true">

#### Question #5

How does the style attribute differ from using an external stylesheet?

* The style attribute applies CSS rules directly to an element, while an external stylesheet requires a separate file.
* The style attribute is used for custom data attributes, while an external stylesheet applies CSS rules directly to an element.
* The style attribute groups multiple elements under the same identifier, while an external stylesheet assigns a unique identifier to an element.
* The style attribute is used to assign a unique identifier to an element, while an external stylesheet provides additional information about an element.

#### Question #6

Which attribute is used to assign a unique identifier to an element?

* style
* class
* id
* data

#### Question #7

Which HTML tag is used to represent a self-contained piece of content that makes sense on its own, such as a blog post or news article?

* <section>
* <article>
* <div>
* <header>

#### Question #8

What is the purpose of the class attribute in HTML?

* To assign a unique identifier to an element.
* To group multiple elements under the same identifier.
* To apply CSS rules directly to an element.
* To store extra information with an element without any visual representation.

#### Question #9

What is the role of the character set in HTML metadata?

* It specifies the language in which the webpage content is written.
* It determines the font styles and sizes used on the webpage.
* It defines the color scheme and design elements of the webpage.
* It selects the alphabet used for encoding text characters on the webpage.

## **Please make sure to validate all quiz questions before moving on to project tasks**

# TEST

## **Learning Objectives**

At the end of this project, you are expected to be able to [explain to anyone](https://intranet.alxswe.com/rltoken/wMEI3Uq9YrfXo98kxNWU0w), **without the help of Google**:

### General

* What is HTML
* How to create an HTML page
* What is a markup language
* What is the DOM
* What is an element / tag
* What is an attribute
* How does the browser load a webpage

## **Requirements**

### General

* Recommended editors: Visual studio code
* All your files should end with a new line
* A README.md file, at the root of the folder of the project is mandatory
* You are **not allowed** to install, import or use external libraries. This website must be build with only HTML/CSS/JavaScript. No NodeJS, React, VueJS, Bootstrap, etc.
* Your code should be W3C compliant and validate with [W3C-Validator](https://intranet.alxswe.com/rltoken/k_qWagKGYKVC6FWS8E_vJQ)

### Quiz questions

**Great!** You've completed the quiz successfully! Keep going!(Show quiz)

## **Tasks**

### 0. Basic Structure

**mandatory**

Create the most basic, simple and valid HTML file called base\_index.html

The page should contain:

* a DOCTYPE
* a head:
  + with a title
* a body

**Tips:** don’t forget to read all concept pages…

Test the webpage with the [official HTML Validator](https://intranet.alxswe.com/rltoken/cBcUPTeOAn3QF4ZReawnjA). You will see that it is not happy!

First things first, fix the code in base\_index.html so that it is valid.

You will see that the validator then becomes much happier!

**Repo:**

* GitHub repository: My\_First\_Portfolio
* Directory: html\_basic
* File: base\_index.html

### 1. Make a cool first webpage

**mandatory**

Copy your file base\_index.html to index.html

Add in the body of the page:

* at least 4 paragraphs;
* headings (titles) of level 1, 2 and 3, which are represented by <h1>, <h2> and <h3> tags;
* a clickable image of your choice (you can use this [link](https://i.ibb.co/gTDZZT8/ALX-Logo-07.png)https://i.ibb.co/gTDZZT8/ALX-Logo-07.png image, or another one you find on the internet by using its full URL); and it may link to any URL online that you want.

**Repo:**

* GitHub repository: My\_First\_Portfolio
* Directory: html\_basic
* File: index.html

### 2. Make a cool other webpage

**mandatory**

Copy your file base\_index.html to tweets.html

Add in the body of the page:

* embed at least one tweet (Here’s the official documentation from Twitter about [embedding single tweets](https://intranet.alxswe.com/rltoken/rM0nwiMGbeSmO7S-MVbIrQ) in a webpage’s HTML code.)
* contain a link to your index.html webpage, for users who may go back to your homepage

In index.html, add a link to tweets.html, for users landing on your homepage, who want to see your tweets!

**Repo:**

* GitHub repository: My\_First\_Portfolio
* Directory: html\_basic
* File: index.html, tweets.html

### 3. Make a cool website

**mandatory**

However, in order to start getting consistency throughout the website, all of the HTML files (index.html and tweets.html) must now have the same structure:

* The <body> tag of all of your webpages must contain exactly three direct sub-tags in that order: <header>, <main> and <footer>
* The <header> tag must contain an unordered list (the <ul> and <li> tags) of links (the <a> tag) to each of your webpages. As you may have understood, this will serve as a navigation for your website.
* The <footer> tag must contain one paragraph with the sentence Made by <YOUR NAME> - <a href="<ANY LINK>" target="\_blank">here</a>.
* The <main> tag must contain two direct sub-tags in that order: <article> and <aside>:
  + <article> contains the content of your webpage: texts, links, images, tweets, … This is the part of index.html and tweets.html that you have already written.
  + <aside> contains a single paragraph, just reading for now “placeholder to add comment thread later”.

**Repo:**

* GitHub repository: My\_First\_Portfolio
* Directory: html\_basic
* File: index.html, tweets.html

### 4. Add some level of creativity

**mandatory**

Update your index.html to add a table with information about some of the learners.

On a new line before the last </article> tag, let’s introduce a container for the table of learners.

The bottom part of the <article> tag (before its closing tag) should contain a table <table> with the following structure:

* Use the <thead> tag to define the table header row.
* Use the <tbody> tag to contain the table data rows.
* Populate the table with the following data:

|  |  |  |
| --- | --- | --- |
| **Name** | **Age** | **Country** |
| John | 25 | USA |
| Emily | 28 | Canada |
| Michael | 32 | Australia |
| Sophia | 21 | UK |

**Repo:**

* GitHub repository: My\_First\_Portfolio
* Directory: html\_basic
* File: index.html

### 5. Build a Personal Portfolio Webpage

**mandatory**

#### Objective

Create a personal portfolio webpage that effectively showcases your skills, projects, and contact information.

#### Detailed Requirements

**Create a Github repository named My\_First\_Portfolio if not already created and within it create a directory called portfolio**

1. **HTML File**:
   * Create an HTML file within the portfolio directory and name your HTML file index.html.
2. **Page Structure and Semantic HTML**:
   * The document must start with the <!DOCTYPE html>declaration.
   * Use <html lang="en"> to define the document language.
   * The <head> section must contain <meta charset="utf-8"> and a <title> element with your full name followed by - Personal Portfolio(e.g., John Doe - Personal Portfolio).
   * The body of the document must include the following semantic HTML5 elements in order: <header>, <nav>, <section>, <article>, and <footer>.
3. **Header**:
   * The <header> must contain an <h1> element with your name.
   * Include a <p> tag beneath the <h1> with a brief tagline or professional statement.
4. **Navigation Bar (<nav>)**:
   * Include a list of links (<ul> containing <li>elements) that allows navigation to the different sections of the webpage: Introduction, Projects, About Me, and Contact.
   * Each list item must contain an <a> tag with the hrefattribute pointing to the corresponding section ID (e.g., href="#projects" for the Projects section).
5. **Content Sections**:
   * **Introduction Section (<section id="introduction">)**: Include an <h2> heading titled “Introduction” and a paragraph <p> describing yourself.
   * **Projects Section (<section id="projects">)**: Use an <h2> heading titled “Projects”. Each project must be wrapped in <article> tags with a class of “project”. Each <article> must contain an <h3> for the project title, a <p> for the description, and an <a> tag linking to the project with target="\_blank".
   * **About Me Section (<section id="about">)**: Include an <h2> heading titled “About Me” and paragraphs or lists describing your background, skills, or hobbies.
   * **Contact Section (<section id="contact">)**: Use an <h2> heading titled “Contact” and provide an email link using <a href="mailto:your.email@example.com"> and links to your LinkedIn and GitHub profiles, ensuring these links open in a new tab using target="\_blank".
6. **Footer**:
   * The <footer> must contain a <p> element with copyright information (e.g., &copy; 2024 Your Name).

**Repo:**

* GitHub repository: My\_First\_Portfolio
* Directory: portfolio
* File: index.html

**Zzzzzzzzzzzzzzzzzzzzzzz**

This project challenges you to build a comprehensive website for the TechCon 2024 conference using HTML. You’ll create various webpages, each focusing on a specific aspect of the conference, while incorporating essential HTML elements and best practices.

### Project Scope:

The website will consist of five core webpages:

1. Homepage (index.html): This is the landing page that introduces TechCon 2024 and provides navigation to other sections of the website.
2. About Page (about.html): This page delves into the history, mission, and notable speakers of the TechCon conference.
3. Schedule Page (schedule.html): This webpage presents a detailed schedule of conference events, sessions, and speakers in an accessible table format.
4. Register Page (register.html): This page offers a user-friendly registration form for attendees to sign up for the TechCon conference.
5. Contact Page (contact.html): This webpage provides comprehensive contact information for TechCon, including email address, social media links, an embedded map, and a “Contact Us” form.

### Concepts Covered:

Throughout this project, you’ll gain practical experience with the following HTML concepts:

* **HTML Structure:** You’ll solidify your understanding of fundamental HTML tags like <header>, <nav>, <main>, <section>, <article>, <footer>, and their proper usage for website structure and organization.
* **Headings and Text:** You’ll learn to structure headings (h1-h6) for emphasis and utilize paragraph tags (<p>) and other text formatting elements effectively.
* **Links and Navigation:** You’ll implement hyperlinks (<a>tags) to create navigation menus and link between different webpages within the TechCon website.
* **Tables:** You’ll create a well-formatted table on the Schedule page to present conference schedules in an organized and easy-to-read manner.
* **Forms:** You’ll design user-friendly registration and contact forms using the <form> tag and various input types (text, email, password, checkbox) to collect attendee information.
* **Images and Multimedia:** You’ll incorporate images (using <img> tags with appropriate alt attributes) to enhance the visual appeal of the website. You’ll also explore embedding a promotional video (using the <video> tag) and an interactive Google Map (using <iframe>) on the Contact page.

By successfully completing this project, you’ll gain valuable hands-on experience building a real-world website with HTML. This will serve as a solid foundation for you to explore more advanced web development concepts like CSS and JavaScript in the future.