माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकुद मन्त्रालय

ICT BOOTCAMP MANUAL ON

Web Development and Cybersecurity

GOVERNMENT OF GANDAKI PROVINCE
MINISTRY OF SOCIAL DEVELOPMENT, YOUTH AND SPORT
Pokhara

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकूद मन्त्रालय

Table of Contents

माननीय विन्दु कुमार धापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकूद मन्त्रालय

सचिव

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकुद मन्त्रालय

Web Development and Cybersecurity

ICT BOOTCAMP

1. Duration: 10 Days (6 hours per day)

2. Goal: To provide the students with the knowledge of the fundamentals of web development, leading to the creation and deployment of a responsive, interactive website with awareness of cyber security.

3. **Syllabus** (Unit Content, Objective and Hands-on Activities)

Day 1	• **Unit Objective:** ○ Understand how the web works and the role of web development. • **Activities:** ○ Set up a folder structure for web projects. ○ Create your first "Hello, World!" HTML file.	**1. Introduction to Web Development** 1.1. Overview of Web Development 1.1.1. Frontend 1.1.2. Backend 1.1.3. Full-Stack 1.2. How the internet works 1.2.1. HTTP 1.2.2. DNS 1.2.3. Browsers 1.2.4. Servers 1.3. Setting UP the Development Environment 1.3.1. Installing a Code Editor (VS Code) 1.3.2. Introduction to developer tools in a browser 1.4. Cybersecurity and its Need 1.5. Insecure vs. Secure Websites
Day 2	• **Unit Objective:** ○ Learn the structure and semantics of a webpage • **Activities:** ○ Create a basic webpage with headings, paragraphs, images, and links.	**2. Introduction to HTML** 2.1. Basic HTML Structure 2.1.1. Basic HTML Tags: <html>, <head>, <body> 2.1.2. Text: <p>, <h1> to <h6> 2.1.3. Links: <a> 2.1.4. Images: <img> 2.1.5. Lists: , , 2.1.6. Table tags 2.2. Semantic HTML

		2.2.1. <header> 2.2.2. <footer> 2.2.3. <article> 2.2.4. <section>
Day 3	• **Unit Objective:** ○ Learn how to style web pages using CSS. • **Activities:** ○ Style the web page created on Day 2 using CSS to improve its appearance.	**3. Introduction to CSS** 3.1. Adding CSS to HTML 3.1.1. Inline 3.1.2. Internal 3.1.3. External CSS 3.2. CSS Basics 3.2.1. Colors, Fonts, and Text Styling 3.2.2. Margins, Padding, and Borders 3.3. Understanding the Box Model
Day 4	• **Unit Objective:** ○ Learn how to create responsive layouts. • **Activities:** ○ Build a responsive web page layout using Flexbox and media queries.	**4. CSS Layout and Responsive Design** 4.1. CSS Layout Techniques 4.1.1. Flexbox: Aligning and distributing items. 4.1.2. Grids (basic introduction) 4.2. Introduction to Media Queries 4.2.1. Adapting layouts for mobile screens 4.2.2. Adapting layouts for desktop screens
Day 5	• **Unit Objective:** ○ Add interactivity to web pages using JavaScript • **Activities:** ○ Create a button that changes the text or color of an element when clicked.	**5. Introduction to JavaScript** 5.1. JavaScript Basics: 5.1.1. Variables, Data Types, and Operators 5.2. Functions and Event Handling Primitive Data Types in Java script Reference Data Types in Java Script
Day 6	• **Unit Objective:** ○ Deepen JavaScript knowledge and explore more interactive capabilities. • **Activities:** ○ Create a simple form with JavaScript validation.	**6. Advanced JavaScript** 6.1. JavaScript Arrays and Objects 6.2. Loops and Conditional Statements 6.3. Introduction to the DOM (Document Object Model) 6.4. DOM Manipulation Techniques 6.4.1. Adding and Removing Elements 6.4.2. Modifying Attributes and Styles

Day 7	• **Unit Objective:** ○ Combine JavaScript and CSS to enhance user interaction. • **Activities:** ○ Create a navigation menu that highlights the active section. ○ Add animations to buttons or images on hover/click.	**7. Integrating JavaScript and CSS** 7.1. Adding CSS Classes Dynamically 7.2. Introduction to Basic Animations with JavaScript 7.3. Event Listeners and User Feedback
Day 8	• **Unit Objective:** ○ Learn how to use version control and deploy a website. • **Activities:** ○ Push a project to GitHub and host it using GitHub Pages.	**8. Introduction to Git and Hosting** 8.1. Introduction to Git 8.1.1. Setting up Git 8.1.2. Basic Commands: init, add, commit, push 8.2. Introduction to GitHub 8.2.1. Creating a repository 8.2.2. Pushing and hosting a website on GitHub Pages
Day 9	• **Unit Objective:** ○ Apply all learned skills to build a complete, responsive, and interactive website. • **Activities:** ○ Students work on their final projects.	**9. Building a Final Project** 9.1. Project Planning 9.1.1. Define the structure and content 9.2. Development 9.2.1. Use HTML, CSS, and JavaScript to build the website 9.2.2. Using Excel for Data analysis in Project Planning 9.3. Hands-On Security Testing 9.3.1. Introduction to tools like OWASP ZAP 9.3.2. Test for vulnerabilities: XSS, SQL Injection, insecure cookies.
Day 10	• **Unit Objective:** ○ Deploy and present the final project. • **Activities:** ○ Present the final project to the group. ○ Receive feedback and discuss improvements	**10. Final Project Deployment and Showcase** 10.1. Final Deployment 10.1.1. Review the project 10.1.2. Fix any bugs or issues 10.2. Presenting the Project 10.3. Q&A and Feedback 10.4. Evaluation and Certification

Chapter 1

Introduction to Web Development

Chapter Summary:

This chapter focuses on creating, building and maintaining websites along with aspects such as browser basics, internet and security. It also deals with the idea of the working mechanism of the internet and other important aspects of web management.

Learning Objectives:

- The goal of the front end development is to create a website's user interface and visual component that users may interact with directly.
- The goal of the back end development works with databases, server side logic and application functionality.
- It enables the users to interact with the content and perform specific tasks.

1.1 Overview of Web Development

Web Development involves creating and maintaining websites and web applications.

1.1.1. Frontend

Focuses on the visual and interactive aspects of a website using technologies like HTML, CSS, and JavaScript. This is what users see and interact with.

1.1.2. Backend

Handles the server-side logic, databases, and application functionality using languages like Python, PHP, Java, or Node.js. It ensures data processing and communication between the server and frontend.

1.1.3. Full-Stack

Combines both frontend and backend development, managing the entire web development process.

1.2 How the internet works

The Internet is a vast network that connects millions of private, public, academic, business, and government networks worldwide, allowing devices to communicate and share information. The Internet is a vast, sprawling collection of networks that connect to each other. In fact, the word "Internet" could be said to come from this concept: interconnected networks. This makes it possible to rapidly exchange information between computers across the world.

- Computers connect to each other and to the Internet via wires, cables, radio waves, and other types of networking infrastructure.
- All data sent over the Internet is translated into pulses of light or electricity, also called "bits," and then interpreted by the receiving computer.
- The wires, cables, and radio waves conduct these bits at the speed of light. The more bits that can pass over these wires and cables at once, the faster the Internet works.

1.3 HTTP

HTTP stands for "Hypertext Transfer Protocol." It is a set of rules for sharing data on the World Wide Web(WWW). HTTP helps web browsers and servers communicate, allowing people to access and share information over the internet.

When you visit a website, HTTP helps your browser request and receive the data needed to display the web pages you see. It is a fundamental part of how the internet works, making it possible for us to browse and interact with websites.

1.4 DNS

In web development, DNS (Domain Name System) is a foundational technology that translates human-readable domain names (like www.example.com) into IP addresses (like 192.168.1.1) that computers use to identify each other on the network. It acts as the "phonebook" of the internet.

1.5 Browsers

A **browser** (short for web browser) is a software application that allows users to access, retrieve, and display content from the World Wide Web. Browsers serve as the interface between users and websites, interpreting web technologies like HTML, CSS, and JavaScript to render web pages visually.

1.6 Servers

A **server** is a computer or system that provides resources, services, or data to other devices, called **clients** (such as web browsers or mobile apps), over a network. The server processes requests from clients and delivers the requested data, such as a web page, file, or application.

Setting UP the Development Environment

Setting up a development environment for web development involves installing tools, configuring your system, and organizing workflows to efficiently build, test, and deploy websites or web applications. The following tasks are important.

1. Choose your development stack
2. Install essential tools

3. Set up front end development environment
4. Set up backend development environment
5. Install supporting tools
6. Configure a local web server
7. Setup database if needed
8. Setup Testing tools
9. Configure your environment

1.7 Installing a Code Editor (VS Code)

Visual Studio Code is the most popular code editor and the IDEs provided by Microsoft for writing different programs and languages. It allows the users to develop new code bases for their applications and allow them to successfully optimize them. Quick preview to install Visual Studio Code on windows:

1. Download the VS Code file from the Official Website.
2. Execute the download file.
3. Accept the terms and conditions.
4. Click on the Install button.
5. Wait for the installation to complete.
6. Click on the launch button to start it.

1.8 Introduction to developer tools in a browser

Developer Tools (DevTools) in a browser are built-in utilities that allow developers to inspect, debug, and analyze web pages and applications. They are essential for understanding how a website works and for troubleshooting problems in your code.

How to Access Developer Tools?

1.Shortcut Keys:

Chrome/Edge: Press Ctrl+Shift+I (Windows/Linux) or Cmd+Option+I (Mac).

Firefox: Press Ctrl+Shift+K (Windows/Linux) or Cmd+Option+K (Mac).

2.Right-click:

Right-click on any element on a webpage and select Inspect or Inspect Element.

3.Menu Navigation:

Open the browser menu → More Tools → Developer Tools.

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकुद मन्त्रालय

Cybersecurity and its Need

Cybersecurity refers to the practice of protecting computer systems, networks, and digital data from unauthorized access, theft, damage, or disruption. It involves a combination of technologies, processes, and practices designed to secure systems and mitigate threats.

Their importance are in various fields like below:

1. Protection of sensitive information.
2. Growing cyber threats.
3. Critical Infrastructure
4. Financial Loss
5. Increased Connectivity.

Insecure vs. Secure Websites

Emphasize why website security matters for individuals and organizations, such as protecting sensitive data and maintaining trust.

1. **Secured websites**: Websites that use security measures like encryption, authentication, and secure communication protocols to protect user data.
2. **Unsecured websites:** Websites that lack proper security measures, making them vulnerable to attacks or data theft

सचिव

Chapter 2

Introduction to HTML

Chapter Summary:

HTML (HyperText Markup Language) is the standard language used to create and structure web pages. It is essential for building websites and serves as the foundation of web development.

Learning Objectives:

- It is essential for building websites and serves as the foundation of web development.
- Allow Developers to structure and present content effectively.
- A solid understanding of HTML is a prerequisite for learning more advanced web technologies like CSS and JavaScript.

2.1. Basic HTML Structure

HTML stands for Hyper Text Markup Language and is the standard markup language used for creating web pages. It describes the structure of a webpage and consists of a series of elements. These elements guide the browser on how to display content on the page. HTML elements are used to label parts of the content, such as headings, paragraphs, and links. HTML provides the foundation for building and organizing web content effectively.

Simple HTML Structure:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

2.1.1. Basic HTML Tags: <html>, <head>, <body>

1.<html> Tag

The <html> tag represents the root element of an HTML document.It encompasses all the other HTML elements and indicates the beginning and end of the document structure

```
<!DOCTYPE html>
<html>
<!-- Content goes here -->
</html>
```

2.<head> Tag

The <head> tag contains metadata and information about the document that is not directly displayed on the web page.

Common elements inside <head>:

1. <title>: Specifies the title of the web page displayed on the browser tab.
2. <meta>: Provides metadata like character encoding, viewport settings, and SEO keywords.
3. <link>: Links to external resources, like CSS stylesheets.
4. <script>: Links or embeds JavaScript files.

Example:

```
<head>
<title>Page Title</title>
<meta charset="UTF-8">
<link rel="stylesheet" href="styles.css">
</head>
```

3.<body> Tag

The <body> tag contains the main content of the HTML document that is visible to users in the browser.All text, images, videos, forms, and other content elements are placed inside the <body> tag.

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकुद मन्त्रालय

Example:

```
<body>
 <h1>Welcome to My Website</h1>
 <p>This is a paragraph.</p>
 <img src="image.jpg" alt="Example Image">
</body>
```

2.1.2. Text: <p>, <h1> to <h6>

1.<p> Tag

The <p> tag is used to define a paragraph in HTML.It automatically adds spacing before and after the paragraph for better readability.

Syntax:

```
<p>This is a paragraph of text.</p>
```

2. <h1> to <h6> Tags

These are heading tags used to define headings of different levels.<h1> is the largest and most important heading, while <h6> is the smallest and least important.Used for organizing content and establishing a hierarchy on a webpage.

Syntax:

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```

2.1.3 Links: <a>

Links: <a> Tag

The <a> tag, also known as the anchor tag, is used to create hyperlinks in HTML.. It allows users to navigate to other pages, sections within the same page, or external resources.

Syntax:

```
<a href="URL">Link Text</a>
```

href Attribute: Specifies the destination URL for the link.

Link Text: The clickable part of the hyperlink, visible to the user.

2.1.4 Images: <img>

Images: <img> Tag

The <img> tag is used in HTML to embed images into a webpage. It is a self-closing tag, meaning it does not require a closing tag. Images enhance the visual appeal and functionality of a website.

Syntax:

<img src="imageURL" alt="Description of the image">

src: The URL or path of the image file. This attribute is required.

alt: Alternative text describing the image. It is displayed if the image cannot load and improves accessibility for screen readers.

2.1.5 Lists: <ul>, <ol>,

Lists: <ul>, <ol>,

HTML lists are used to group and organize related items, making them easier to read and navigate. The primary types of lists are unordered lists (<ul>), ordered lists (<ol>), and the list items they contain (<li>).

**1. Unordered List: **

- Displays items as a bulleted list.
- Used when the order of items is not important.

Syntax:

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

**2. Ordered List: **

Displays items in a numbered list.Used when the order of items is important (e.g., steps in a process).

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकूद मन्त्रालय

Syntax:

```
<ol>
<li>Step 1</li>
<li>Step 2</li>
<li>Step 3</li>
</ol>
```

**3. List Item: **

The <li> tag represents a single item within a list.Can be used in both <ul> and <ol>.

Example:

```
<ul>
<li>First item</li>
<li>Second item</li>
</ul>
```

2.1.6 Table tags

HTML Table Tags

HTML tables are used to display tabular data in rows and columns. The table structure is defined using several specific tags.

Key Table Tags and Their Functions

1. **<table>**
   - Defines the table container.
   - All table content, including rows and columns, is placed inside this tag.
2. **<tr> (Table Row)**
   - Defines a row in the table.
   - Contains one or more cells (<td> or <th>).
3. **<td> (Table Data)**
   - Represents a standard data cell in a table.

- Used for regular table content.

4. **<th> (Table Header)**
   - Defines a header cell.
   - Typically displayed in bold and centered by default.
5. **<thead> (Table Head)**
   - Groups the header rows of a table.
6. **<tbody> (Table Body)**
   - Groups the main content rows of a table.
7. **<tfoot> (Table Footer)**
   - Groups the footer rows of a table.
   - Useful for summarizing data or displaying totals.
8. **<caption>**

   Provides a title or description for the table.

   Syntax:

```
<table>
<tr>
 <th>Header 1</th>
 <th>Header 2</th>
 <th>Header 3</th>
</tr>
<tr>
 <td>Data 1</td>
 <td>Data 2</td>
 <td>Data 3</td>
</tr>
<tr>
 <td>Data 4</td>
 <td>Data 5</td>
 <td>Data 6</td>
</tr>      </table>
```

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकुद मन्त्रालय

2.1.7 Semantic HTML

Semantic HTML refers to the use of HTML tags that have meaningful names, describing the content they enclose. These tags improve accessibility, SEO, and readability for both developers and browsers.

2.1.8 <header>

Represents the header section of a page or content area.Often includes logos, navigation, or introductory content.

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

2.1.9 <footer>

Represents the footer of a section or page, often used for copyright or contact information.

Syntax:

```
<footer>
<p>&copy; 2024 My Website</p>
<a href="/privacy">Privacy Policy</a>
</footer>
```

सचिव

2.1.10 <article>

Represents a self-contained piece of content (e.g., a blog post, news article).

Syntax:

```
<article>
<h2>Breaking News</h2>
<p>Today, a new technology was unveiled...</p>
</article>
```

2.1.11 <section>

Groups related content together within a webpage.

Syntax:

```
<section>
<h2>Our Services</h2>
<p>We offer web design and development services.</p>
</section>
```

Chapter 3

Introduction to CSS

Chapter Summary:

Cascading Style Sheets (CSS) is used to format the layout of a webpage. With CSS, you can control the color, font, the size of text, the spacing between elements, how elements are positioned and laid out, what background images or background colors are to be used, different displays for different devices and screen sizes, and much more.

Learning Objectives:

- Style elements by defining properties like color, font, size, spacing, and positioning.
- Keeps design separate from the content, making it easier to maintain and update websites.
- Creation of responsive layouts that adapt to different devices and screen sizes.

3.1 Adding CSS to HTML

There are three main ways to add CSS to an HTML document: **Inline CSS**, **Internal CSS**, and **External CSS**. Each method has its specific use cases and benefits.

Inline

Inline CSS is applied directly within an HTML element using the style attribute. It is used for quick, one-off styling of a specific element.

Syntax:

```
<tagname style="property: value;">
Content
</tagname>
```

Internal

Internal CSS is placed inside the <style> tag in the <head> section of an HTML document. It is useful for styling a single HTML page.

Syntax:

```
<head>
<style>
 selector {
  property: value;
```

```
}
</style>
</head>
```

- **External CSS**

External CSS involves linking to an external stylesheet that contains all the CSS rules. This method is the most efficient for styling multiple pages consistently.

Syntax:

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

CSS Basics

CSS offers a wide variety of properties to control the color, typography, and appearance of text on web pages

Colors, Fonts, and Text Styling

CSS provides several ways to define colors for text, backgrounds, borders, and more. Here are the main methods to specify colors:

Named Colors

You can use predefined color names in CSS. For example, red, blue, green, etc.

Syntax:

```
h1 {
 color: blue;
}
```

Hexadecimal Colors

Hex codes are a 6-digit combination of numbers and letters (from #000000 to #FFFFFF) representing colors.

Syntax:

```
body {
  background-color: #f0f0f0; /* Light gray background */
```

माननीय विधु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकुद मन्त्रालय

}

RGB Colors

RGB (Red, Green, Blue) colors are defined by specifying the intensity of red, green, and blue light (values range from 0 to 255).

Syntax:

```
p {
  color: rgb(255, 87, 51); /* RGB color for a red-orange */
}
```

HSL Colors

HSL stands for **Hue**, **Saturation**, and **Lightness**. It defines colors using a hue angle (0 to 360), saturation (0% to 100%), and lightness (0% to 100%).

Syntax:

```
h2 {
  color: hsl(120, 100%, 50%); /* Green color */
}
```

Fonts in CSS

CSS allows you to specify custom fonts for text on a webpage. You can define font properties like font family, font size, font weight, and more.

Font Family

The font-family property defines the type of font used for the text. You can specify multiple font families, and the browser will use the first one available.

Syntax:

```
body {
  font-family: 'Arial', sans-serif;
}
```

सचिव

Font Size

The font-size property defines the size of the text. You can specify the size in units like pixels (px), ems (em), percentages (%), or rems (rem).

Syntax:

```
p {
  font-size: 16px; /* 16 pixels */
}
```

Font Weight

The font-weight property controls the thickness of the text. You can use keywords like normal, bold, or specify a numeric value between 100 (thin) and 900 (bold).

Syntax:

```
h1 {
 font-weight: bold;
}
```

Font Style

The font-style property defines whether the text is italicized or normal.

Syntax:

```
em {
 font-style: italic;
}
```

Font-weight

Defines the weight (thickness) of the font. Common values are normal, bold, lighter, or numbers (100–900).

Syntax:

```
strong {
  font-weight: bold;
}
```

Margins, Padding, and Borders

1.Margin: It creates space outside the element, separating it from other elements.

Properties:

- margin: Shorthand for all sides.
- margin-top, margin-right, margin-bottom, margin-left: Control individual sides.

Example:

```
div {
  margin: 20px; /* Adds 20px space around the element */
  margin-top: 10px;
  margin-left: auto; /* Center the element horizontally (when combined with a fixed width) */
}
```

2.Padding: It creates space inside the element, between the content and the border.

Properties:

- padding: Shorthand for all sides.
- padding-top, padding-right, padding-bottom, padding-left: Control individual sides.

Example:

```
div {
  padding: 15px; /* Adds 15px inside the element */
  padding-right: 20px;
}
```

3.Border

What it does: Adds a line around the element. Borders sit between the margin and padding.

Properties:

- border: Shorthand for width, style, and color.
- border-width, border-style, border-color: Control each aspect individually.
- border-top, border-right, border-bottom, border-left: Control individual sides.
- Common Border Styles: solid, dashed, dotted, double, none.

Example:

```
div {
border: 2px solid black; /* 2px solid black border */
border-radius: 10px; /* Rounds the corners */
}
```

Understanding the Box Model

The CSS Box Model is a fundamental concept in web design, as it defines how elements are structured and how their sizes are calculated. Every HTML element is represented as a rectangular box, consisting of the following parts:

Parts of the Box Model

1. **Content:**
   - The actual content of the element (e.g., text, images, or other elements).
   - Defined by properties like width and height.
2. **Padding**:
   - The space between the content and the border.
   - Adds "breathing room" around the content.
   - Transparent and takes up space.
3. **Border:**
   - A line wrapping the padding and content.
   - Its size and style can be customized using border-width, border-style, and border-color.
4. **Margin:**
   - The space outside the border that separates the element from other elements.
   - Transparent and collapses with other margins when adjacent.

Box Model Structure

```
+---------------------------+
|     Margin (outside)    |
| +---------------------+ |
| |    Border         | |
| | +---------------+ | |
| | | Padding     | | |
| | |+-----------+ | | |
```

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकूद मन्त्रालय

```
| | || Content | | | |
| | |+----------+ | | |
| | +--------------+ | |
| +--------------------+ |
+------------------------+
```

Visualizing Sizes

The total size of a box is calculated as:

Total Width = Content Width + Padding (left + right) + Border (left + right) + Margin (left + right)
Total Height = Content Height + Padding (top + bottom) + Border (top + bottom) + Margin (top + bottom)

Example:

```
div {
    width: 200px;          /* Content width */
    height: 100px;         /* Content height */
    padding: 20px;         /* Space inside the box */
    border: 5px solid black; /* Border width */
    margin: 10px;          /* Space outside the box */
}
```

- Content width: 200px
- Total width: 200px + 20px (padding-left) + 20px (padding-right) + 5px (border-left) + 5px (border-right) + 10px (margin-left) + 10px (margin-right) = 270px

Box-Sizing Property

The box-sizing property determines how the box's total size is calculated:

1. **Default** (content-box):
    - Only the content dimensions are included in the width and height.

- Padding and border are added outside the content area.

Syntax:

```
div {
box-sizing: content-box; /* Default */
}
```

2.**border-box:**

- Padding and border are included within the specified width and height.
- This makes it easier to manage layouts.

Syntax:

```
div {
box-sizing: border-box;
}
```

Chapter 4

CSS Layout and Responsive Design

Chapter Summary:

This chapter focuses on how to create flexible and responsive layouts using modern CSS techniques like Flexbox and Grid. It introduces media queries to adapt designs for different screen sizes.

Learning Objectives:

- Understand CSS layout techniques (Flexbox and Grid).
- Learn how to make layouts responsive using media queries.
- Build layouts that adapt to various screen sizes like mobile and desktop.

4.1. CSS Layout Techniques

4.1.1. Flexbox: Aligning and distributing items.

What is Flexbox?

Flexbox is a one-dimensional layout method for aligning and distributing space among items in a container.

How to Use Flexbox?

Example Code:

```
<div style="display: flex; justify-content: center; align-items: center; height: 100vh;">
  <div>Item 1</div>
  <div>Item 2</div>
  <div>Item 3</div>
</div>
```

Output: Items are aligned both horizontally and vertically.

4.1.2. Grids (basic introduction)

CSS Grid is a two-dimensional layout system. It allows for precise placement of items in rows and columns.

Example Code:

```
<div style="display: grid; grid-template-columns: repeat(3, 1fr); gap: 10px;">
  <div>Item 1</div>
```

```
    <div>Item 2</div>
    <div>Item 3</div>
</div>
```

4.2. Introduction to Media Queries

Media queries are a CSS feature that allows you to apply styles based on the characteristics of the user's device, such as screen size, resolution, or orientation. They are commonly used to make web designs responsive.

4.2.1. Adapting layouts for mobile screens

Example Code:

```
@media (max-width: 768px) {
  body {
    font-size: 14px;
  }
}
```

4.2.2. Adapting layouts for desktop screens

Example Code:

```
@media (min-width: 769px) {
  body {
    font-size: 18px;
  }
}
```

Chapter 5

Introduction to JavaScript

Chapter Summary:

This chapter introduces JavaScript basics, covering variables, data types, and the fundamentals of event handling and DOM(Document Object Model) manipulation.

Learning Objectives:

- Learn JavaScript basics like variables and data types.
- Understand functions and how to handle events.
- Gain a basic understanding of the DOM and its manipulation.

5.1. JavaScript Basics

5.1.1. Variables, Data Types, and Operators

Variables:

Variables are used to store data that can be used and manipulated later in a program. In JavaScript, variables can hold values of various data types, such as numbers, strings, and objects.

JavaScript uses three keywords to declare variables:

- var: The old way to declare variables (global or function-scoped).
- let: Introduced in ES6, allows block-level scoping.
- const: Used to declare variables that do not change.

Example: Declaring Variables in JavaScript

```
// Declaring variables using var

var name = "John";

// Declaring variables using let

let age = 25;

// Declaring a constant using const
```

```
const country = "Nepal";

console.log("Name: " + name); // Outputs: Name: John

console.log("Age: " + age);   // Outputs: Age: 25

console.log("Country: " + country); // Outputs: Country: Nepal
```

Data Types:

Data types define the type of data that a variable can hold. In JavaScript, data types are broadly categorized into:

1. **Primitive Types**: Basic, immutable data types directly assigned and stored.
2. **Reference Types**: Complex data types stored as references and mutable.

1. Primitive Data Types in JavaScript:

10. **Number**: Represents numerical values, including integers and floating-point numbers.
11. **String**: Represents a sequence of characters, enclosed in single, double, or backticks.
12. **Boolean**: Represents logical values true or false.
13. **Null**: Represents an explicitly empty or null value.
14. **Undefined**: Represents an uninitialized variable.
15. **Symbol**: Represents unique identifiers .
16. **BigInt**: Represents integers larger than Number.MAX_SAFE_INTEGER

2. Reference Data Types in JavaScript

- **Object**: Represents collections of key-value pairs.
- **Array**: Represents a list-like collection of elements.
- **Function**: Represents executable code blocks.

Primitive data types:

```
let number = 42;          // Number

let text = "Hello World";  // String

let isTrue = true;        // Boolean

let nothing = null;       // Null

let notDefined;           // Undefined
```

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकूद मन्त्रालय

Reference data types

```
let array = [1, 2, 3];     // Array

let object = { name: "Alice", age: 30 }; // Object

console.log(typeof number); // Outputs: number

console.log(typeof text);   // Outputs: string

console.log(typeof isTrue); // Outputs: boolean

console.log(typeof nothing); // Outputs: object (special case in JavaScript)

console.log(typeof notDefined); // Outputs: undefined
```

5.2. Functions and Event Handling

A function in JavaScript is a block of code designed to perform a particular task. Functions are executed when they are invoked (called). They help in code reusability and modular programming.

Types of Functions in JavaScript:

Function Declaration: Defined using the function keyword.
Function Expression: Assigned to a variable as an anonymous function.
Arrow Functions: A concise way of writing functions introduced in ES6.

Example: Functions in JavaScript

```
// Function Declaration

function greet(name) {

   return `Hello, ${name}!`;

}

console.log(greet("Alice")); // Outputs: Hello, Alice!

// Function Expression
```


सचिव

```
const add = function(a, b) {
  return a + b;
};
console.log(add(5, 3)); // Outputs: 8
// Arrow Function
const multiply = (a, b) => a * b;
console.log(multiply(4, 5)); // Outputs: 20
```

Event Handling

Event handling refers to the process of capturing and responding to user actions (events) such as clicks, mouse movements, keystrokes, etc. JavaScript uses **Event Listeners** to detect and handle these events.

Common Events:

1. **Click** (onclick)
2. **Mouse Events** (onmouseover, onmouseout, onmousemove)
3. **Keyboard Events** (onkeydown, onkeyup)
4. **Form Events** (onsubmit, onchange)

Example: Event Handling in JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Event Handling Example</title>
</head>
<body>
  <h1 id="message">Click the button below:</h1>
  <button id="clickButton">Click Me</button>
```

```
<script>
    // Select elements
    const button = document.getElementById("clickButton");
    const message = document.getElementById("message");

    // Add an event listener to handle clicks
    button.addEventListener("click", function() {
      message.textContent = "Button was clicked!";
    });
  </script>
</body>
</html>
```

5.3. Introduction to the DOM (Document Object Model)

The DOM is essential for:

- **Manipulating Content**: Add, remove, or modify HTML elements and text dynamically.
- **Handling Events**: Listen for user interactions like clicks, keypresses, or form submissions.
- **Styling Dynamically**: Update CSS styles or classes for visual changes.

Basic Structure of the DOM Tree:

- **Document Node**: Represents the entire HTML document.
- **Element Nodes**: Represent HTML tags such as , <p>, <h1>.
- **Attribute Nodes**: Represent attributes like id, class, or src.
- **Text Nodes**: Represent the text content inside HTML elements.

Example:

For the HTML <h1 id="title">Hello World</h1>, the DOM representation would look like this:

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकुद मन्त्रालय

Document
└── HTML
    └── BODY
        └── H1 (id="title")
            └── Text: "Hello World"

Example: Accessing and Modifying the DOM

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>DOM Example</title>
</head>
<body>
  <h1 id="heading">Welcome to the DOM</h1>
  <button id="changeText">Change Text</button>
  <script>
    // Access the DOM element
    const heading = document.getElementById("heading");
    const button = document.getElementById("changeText");
    // Modify the DOM dynamically on button click
    button.addEventListener("click", function() {
      heading.textContent = "DOM Manipulation is Awesome!";
    });
  </script>
</body>
</html>
```

Accessing Elements:

- **document.getElementById("heading"): Selects the <h1> element with the id of "heading".**

Modifying Elements:

- **heading.textContent: Changes the text inside the <h1> tag dynamically.**

Handling Events:

- **button.addEventListener: Adds a click event listener to the button, triggering the function to update the DOM.**

Chapter 6

Advanced JavaScript

Chapter Summary:

This chapter delves into advanced JavaScript concepts that are essential for creating dynamic and interactive web applications. Topics include working with arrays and objects, implementing loops and conditional statements, advanced DOM manipulation techniques, and modifying attributes and styles of web elements.

Learning Objectives:

- Work efficiently with JavaScript arrays and objects for data storage and manipulation.
- Implement advanced loops and conditional statements for complex decision-making.
- Master DOM manipulation techniques for dynamically updating the structure and style of web pages.

6.1. JavaScript Arrays and Objects

Definition of Arrays and Objects:

Arrays: Ordered collections of data that can hold multiple values in a single variable.
Objects: Collections of key-value pairs used to store more complex data.

Working with Arrays:

- **Creating an Array:**

```
const fruits = ["Apple", "Banana", "Mango"];
```

Common Methods:

- push(): Add an item to the end of the array.
- pop(): Remove the last item.
- forEach(): Execute a function on each array element.

EXAMPLES:

```
const fruits = ["Apple", "Banana", "Mango"];
```

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकुद मन्त्रालय

```
fruits.push("Orange");

console.log(fruits); // ["Apple", "Banana", "Mango", "Orange"]
```

Working with Objects:

Creating an Object:

```
const car = { brand: "Toyota", model: "Corolla", year: 2022 };
```

Accessing Properties:

```
console.log(car.brand); // "Toyota"
```

6.2 Loops and Conditional Statements

Definition:

Loops: Repeatedly execute a block of code.
Conditional Statements: Perform different actions based on conditions.

Types of Loops:

1.for Loop:

Syntax and Example:

```
for (initialization; condition; increment/decrement) {
  // Code to execute
}

for (let i = 0; i < 5; i++) {
  console.log(i); // Output: 0, 1, 2, 3, 4
}
```

2. while Loop

Syntax and Example:

```
while (condition) {
  // Code to execute
}

let i = 0;
while (i < 5) {
  console.log(i); // Output: 0, 1, 2, 3, 4
  i++;
}
```


सचिव

3. do...while Loop

Syntax and Example:

```
do {
  // Code to execute
} while (condition);

let i = 0;
do {
  console.log(i); // Output: 0, 1, 2, 3, 4
  i++;
} while (i < 5);
```

4. for...of Loop

Syntax and Example (Array):

```
for (variable of iterable) {
  // Code to execute
}

const fruits = ["Apple", "Banana", "Mango"];
for (let fruit of fruits) {
  console.log(fruit); // Output: Apple, Banana, Mango
}
```

Syntax and Example (String):

```
for (variable of iterable) {
  // Code to execute
}

const text = "Hello";
for (let char of text) {
  console.log(char); // Output: H, e, l, l, o
}
```

5. for...in Loop

Syntax and Example (Object):

```
for (key in object) {
  // Code to execute
}
```

```
const car = { brand: "Toyota", model: "Corolla", year: 2022 };
for (let key in car) {
  console.log(`${key}: ${car[key]}`);
  // Output:
  // brand: Toyota
  // model: Corolla
  // year: 2022
}
```

Syntax and Example (Array):

```
for (index in object) {
  // Code to execute
}

const numbers = [10, 20, 30];
for (let index in numbers) {
  console.log(`Index: ${index}, Value: ${numbers[index]}`);
  // Output:
  // Index: 0, Value: 10
  // Index: 1, Value: 20
  // Index: 2, Value: 30
}
```

6. Using break in Loops

Syntax and Example:

```
for (let i = 0; i < 10; i++) {
  if (i === 5) break; // Exits the loop when i equals 5
  console.log(i); // Output: 0, 1, 2, 3, 4
}
```

7. Using continue in Loops

Syntax and Example:

```
for (let i = 0; i < 10; i++) {
  if (i === 5) continue; // Skips the iteration when i equals 5
  console.log(i); // Output: 0, 1, 2, 3, 4, 6, 7, 8, 9
}
```

6.3 DOM Manipulation Techniques

DOM (Document Object Model) manipulation involves accessing and modifying elements and their properties within an HTML document using JavaScript.

6.4 Adding and Removing Elements

1. Adding Elements

You can dynamically create and append elements to the DOM.

Syntax and Example:

```
// Create a new element
const newElement = document.createElement('div');
// Add content to the element
newElement.textContent = "Hello, I'm a new div!";
// Append the element to the DOM
document.body.appendChild(newElement);
```

2. Inserting Elements Before Another Element

Syntax and Example:

```
const parentElement = document.getElementById('parent'); // Target parent element
const newChild = document.createElement('p'); // Create new element
newChild.textContent = "I'm inserted before the first child!";

const firstChild = parentElement.firstElementChild; // Find the first child
parentElement.insertBefore(newChild, firstChild); // Insert before the first child
```

3. Removing Elements

Syntax and Example:

```
const elementToRemove = document.getElementById('removeMe');
elementToRemove.remove(); // Removes the element from the DOM
```

6.5 Modifying Attributes and Styles

1. Modifying Attributes

You can add, remove, or modify attributes of an element.

Syntax and Example:

```
const element = document.getElementById('myElement');

// Set a new attribute
element.setAttribute('class', 'new-class');

// Modify an existing attribute
element.setAttribute('id', 'newId');

// Remove an attribute
element.removeAttribute('class');
```

2. Accessing Attributes

Syntax and Example:

```
const element = document.getElementById('myElement');
const idValue = element.getAttribute('id');
console.log(`The ID is: ${idValue}`); // Output: The ID is: myElement
```

3. Modifying Styles

You can dynamically change the CSS styles of an element using JavaScript.

Syntax and Example:

```
const element = document.getElementById('myElement');

// Modify the style directly
```

```
element.style.color = 'red';
element.style.fontSize = '20px';
element.style.backgroundColor = 'yellow';
// Add a CSS class
element.classList.add('highlight');
// Remove a CSS class
element.classList.remove('highlight');
```

Combining Techniques

Here's a combined example that adds an element, modifies its attributes, and changes its styles:

```
// Create and append a new element
const newDiv = document.createElement('div');
newDiv.textContent = "I am a dynamically added div!";
document.body.appendChild(newDiv);
// Modify its attributes
newDiv.setAttribute('id', 'dynamicDiv');
newDiv.setAttribute('class', 'styledDiv');
// Modify its styles
newDiv.style.color = 'white';
newDiv.style.backgroundColor = 'blue';
newDiv.style.padding = '10px';
newDiv.style.borderRadius = '5px';
```

Chapter 7

Integrating JavaScript and CSS

Chapter Summary:

This chapter explains how JavaScript can work together with CSS to make web pages interactive and visually appealing. We'll cover dynamic class manipulation, creating basic animations, and using event listeners for responsive user feedback.

Learning Objectives:

- Apply CSS classes dynamically to change the appearance of elements.
- Create smooth animations using JavaScript without external libraries.
- Utilize event listeners for interactive and responsive designs.

7.1. Adding CSS Classes Dynamically

What Does This Mean?

Instead of modifying individual styles, you can add or remove pre-defined CSS classes to elements dynamically using JavaScript. This approach is cleaner and reusable.

Syntax and Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <style>
  .highlight {
   background-color: yellow;
   font-weight: bold;
  }
 </style>
</head>
<body>
```

```
<p id="myText">This is some text.</p>
<button id="toggleBtn">Toggle Highlight</button>
<script>
 const textElement = document.getElementById('myText');
 const button = document.getElementById('toggleBtn');
 button.addEventListener('click', () => {
  textElement.classList.toggle('highlight'); // Adds or removes the 'highlight' class
 });
</script>
</body>
</html>
```

What Happens?

When the button is clicked, the highlight class is toggled on the text element, changing its style.

7.2. Introduction to Basic Animations with JavaScript

What Are JavaScript Animations?

JavaScript animations allow you to create smooth transitions and movements without relying on CSS animations or external libraries.

Example: Moving a Box Horizontally

```
<!DOCTYPE html>
<html lang="en">
<head>
 <style>
  #box {
   width: 50px;
   height: 50px;
   background-color: blue;
   position: relative;
  }
 </style>
</head>
```

```
<body>
 <div id="box"></div>
 <button id="animateBtn">Move Box</button>
 <script>
  const box = document.getElementById('box');
  const button = document.getElementById('animateBtn');
  button.addEventListener('click', () => {
   let position = 0; // Starting position
   const interval = setInterval(() => {
    if (position >= 300) {
     clearInterval(interval); // Stop animation when position reaches 300px
    } else {
     position += 5; // Increment position
     box.style.left = position + 'px'; // Update box position
    }
   }, 20); // Update every 20 milliseconds
  });
 </script>
</body>
</html>
```

What Happens?

When you click the button, the blue box moves 300px to the right.

7.3. Event Listeners and User Feedback

What Are Event Listeners?

Event listeners allow you to detect user actions (e.g., clicks, hover, input) and respond accordingly. They are essential for providing interactive feedback.

Example: Changing Button Text on Hover

```
<!DOCTYPE html>
<html lang="en">
<head>
 <style>
  #hoverBtn {
   padding: 10px 20px;
   background-color: green;
   color: white;
   border: none;
   cursor: pointer;
  }
 </style>
</head>
<body>
 <button id="hoverBtn">Hover Over Me</button>
 <script>
  const button = document.getElementById('hoverBtn');
  button.addEventListener('mouseover', () => {
   button.textContent = "You're Hovering!";
  });
  button.addEventListener('mouseout', () => {
   button.textContent = "Hover Over Me";
  });
 </script>
</body>
</html>
```

What Happens?

- When you hover over the button, the text changes to "You're Hovering!"

- When you move your mouse away, the text reverts.

Combining Everything:

Here's an example that combines dynamic classes, animations, and event listeners:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <style>
  .box {
   width: 50px;
   height: 50px;
   background-color: blue;
   position: relative;
   margin-top: 20px;
  }
  .highlight {
   background-color: yellow;
   transform: scale(1.5); /* Enlarges the box */
  }
 </style>
</head>
<body>
 <div class="box" id="myBox"></div>
 <button id="animateAndHighlight">Animate & Highlight</button>
 <script>
  const box = document.getElementById('myBox');
  const button = document.getElementById('animateAndHighlight');
  button.addEventListener('click', () => {
   // Add highlight class
   box.classList.toggle('highlight');
```

```
    // Animate movement
    let position = 0;
    const interval = setInterval(() => {
     if (position >= 300) {
      clearInterval(interval);
     } else {
      position += 5;
      box.style.left = position + 'px';
     }
    }, 20);
   });
  </script>
</body>
</html>
```

What Happens?

- When you click the button, the box moves and toggles the highlight effect simultaneously.

Chapter 8

Introduction to Git and Hosting

Chapter Summary:

This chapter introduces Git, a distributed version control system, and its core features, including tracking changes, branching, and collaboration. You will also learn how to set up Git, use essential commands, and host projects on platforms like GitHub or GitLab. By the end, you'll be ready to manage repositories and collaborate effectively.

Learning Objectives:

- Understand the purpose and benefits of Git.
- Learn the steps to install and configure Git on your system.
- Use basic Git commands such as init, add, commit, and push.
- Host and share projects using platforms like GitHub or GitLab.

8.1. Introduction to Git

Git is a distributed version control system designed to track changes in files and coordinate work among multiple people. Its popularity stems from its speed, flexibility, and widespread adoption in software development.

Key Features of Git:

- **Version Control:** Tracks changes to files over time, enabling rollbacks and history tracking.
- **Branching and Merging:** Allows developers to work on different features or fixes simultaneously.
- **Distributed System:** Each user has a full copy of the repository, ensuring redundancy and offline access.
- **Collaboration:** Facilitates teamwork with tools for resolving conflicts and integrating changes.

8.1.1. Setting up Git

Follow these steps to get started with Git:

1. Install Git

- **Windows:** Download and install Git from git-scm.com.
- **macOS:** Install using Homebrew: brew install git.

- **Linux:** Use your package manager: sudo apt-get install git (Debian/Ubuntu) or sudo yum install git (Fedora).

2. Configure Git

Run the following commands to set up your identity:

Set your name

$ git config --global user.name "Your Name"

Set your email

$ git config --global user.email "your.email@example.com"

Verify configuration

$ git config --list

3. Initialize a Repository

Create a new Git repository or clone an existing one to start tracking files.

8.1.2. Basic Commands: init, add, commit, push

1. git init

Initialize a new Git repository in your project directory.

$ git init

Creates a hidden. git folder to store repository metadata.

2. git add

Adds changes in files to the staging area, preparing them for a commit.

Add a specific file

$ git add filename

Add all changes

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकूद मन्त्रालय

$ git add .

3. git commit

Records changes from the staging area in the repository's history.

$ git commit -m "Descriptive message about changes"

Commits should have meaningful messages describing what has been done.

4. git push

Uploads local repository content to a remote repository (e.g., GitHub).

$ git push origin main

Requires a remote repository and appropriate permissions.

Hosting with Git

Git hosting services like GitHub, GitLab, and Bitbucket provide platforms to store, share, and collaborate on repositories. Here's how to connect your local repository to a remote host:

1. Create a Repository on a Hosting Service
   - Log in to your Git hosting platform.
   - Create a new repository and copy its URL.
2. Link Your Local Repository

   $ git remote add origin https://github.com/username/repo.git

3. Push Your Code

   $ git push -u origin main

The -u flag sets the upstream branch for future git push commands.

सचिव

माननीय विन्दु कुमार पांडा
मन्त्री
सामाजिक विकास, युवा तथा खेलकूद मन्त्रालय

8.2. Introduction to GitHub

GitHub is like a social media platform for programmers. It's built around Git, a tool that lets you track changes to your code over time. GitHub is a website and service that allows you to store your code (in repositories) and collaborate with others on programming projects. It is a web-based platform for version control and collaboration. It allows multiple people to work on projects together, track changes, and maintain a history of their work. GitHub uses Git, a version control system, to manage and store revisions of projects.

Key Features:

- **Version Control**: Keep track of every change made to your project.
- **Collaboration**: Work on code with others using branches and pull requests.
- **Hosting**: Host and share your projects online, including websites.
- **Open Source**: Contribute to or use open-source projects hosted on GitHub.
- **Backup:** Your code is safely stored online.
- **Show off your work:** GitHub is a great way to build a portfolio of your projects.

8.2.1 Creating a repository

A repository (or repo) is like a folder for your project. It contains all the project's files and their history. Here's how you can create one:

Sign Up/Sign In:

Go to GitHub and create an account or log in if you already have one.

Create a New Repository:

- On the GitHub dashboard, click the "+" icon in the top-right corner.
- Select **New repository**.

Set Up the Repository:

1. Give your repository a name (e.g., tech_channel).
2. Add a description (optional).
3. Choose between public (visible to everyone) or private (visible only to you and collaborators).
4. Optionally, initialize the repository with a README file. A README introduces your project.
5. Click **Create repository**.

8.2.2 Pushing and hosting a website on GitHub Pages

Pushing Your Code to GitHub

To get your code on to GitHub, you'll need to use Git on your computer. Here are the basic steps:

- **Install Git**: Download and install Git from git-scm.com.
- **Set Up Git**: Open a terminal or command prompt and configure Git with your name and email:
    - git config --global user.name "Your Name"

      git config --global user.email "youremail@example.com"
- **Clone Your Repository**: On GitHub, find the green "Code" button on your repository page. Copy the URL. Open your terminal or command prompt and use:

git clone <URL> This will download a copy of your repository to your computer.

4. **Add Files**: Put your website files (HTML, CSS, JavaScript, etc.) into the cloned repository folder.
5. **Stage Changes**:

   Use:

   git add . *(to tell Git to track all the new files.)*

6. **Commit**:

   Use:

   git commit -m "My first commit" *-to create a snapshot. (Replace "My first commit" with a short message describing your changes.)*

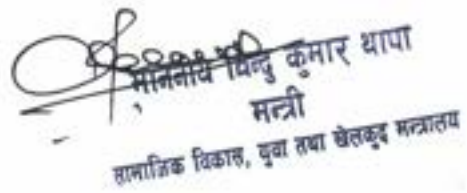
7. **Push**:

   Use:

   git push origin main *(-to send your commit to GitHub.)*

Hosting a Website on GitHub Pages

GitHub Pages is a free way to host your website directly from your GitHub repository.



1. **Prepare Your Website**: Create an index.html file in your project folder. This will be the main page of your website.
2. **Push Your Website Files to GitHub**: Follow the steps above (??) to push your code to a GitHub repository.
3. **Enable GitHub Pages**:
    - Go to your repository on GitHub.
    - Click on Settings.
    - Scroll down to the Pages section.
    - Under "Source", select the branch (e.g., main) and folder (e.g., /root) where your website files are located.
    - Click **Save**.
4. **Access Your Website**: Your website will be live at https://<your-username>.github.io/<repository-name>/.

माननीय विन्दु कुमार थापा
मन्त्री
प्राविधिक शिक्षा, रोजगार तथा खेलकुद मन्त्रालय

Chapter 9

Building a Final Project

Chapter Summary:

Chapter 9 focuses on guiding you through the process of building your final web project, from planning to development, security testing, and deployment. It emphasizes the importance of creating a well-structured project, using appropriate web technologies (HTML, CSS, JavaScript), and integrating security testing to ensure your project is secure before going live.

By the end of this chapter, you will have developed a functional website, tested it for vulnerabilities, and prepared it for deployment. The chapter includes a step-by-step guide to help you plan, develop, and test your website, using tools like OWASP ZAP for security testing and understanding common vulnerabilities such as XSS, SQL Injection, and insecure cookies.

Learning Objectives:

- Plan and design a structured and user-friendly website with essential pages and navigation.
- Develop a responsive and visually appealing site using HTML, CSS, and JavaScript.
- Perform security testing to identify and address vulnerabilities, ensuring a secure and reliable final project.

9.1 Project Planning

Define the structure and content

Start by planning the overall structure of your website. This includes deciding on the key pages and their layout. For instance, a portfolio site could have the following pages:

- Home: Introduction to who you are and what you do.
- About: Information about yourself and your background.
- Projects: Showcase of your work and accomplishments.
- Contact: A form to get in touch with you.

Example:

- **Home Page**: Welcome message, brief introduction, and navigation links to other sections.
- **Projects Page**: Grid of your past projects with links to project details.
- **Contact Page**: Form fields for name, email, message, and a submit button.

Consider User Experience (UX): Think about the flow of the website and ensure it is intuitive. Make sure that visitors can easily navigate through the content.

Content Creation: Create and gather all the content you need for your site (text, images, links). For example, write the text for the "About" page, gather images of your projects, and prepare contact information.

9.2 Development

Use HTML, CSS, and JavaScript to build the website

HTML (Structure):

- Create the website structure using essential HTML tags.
- Organize content into sections like header, main content, and footer.
- Include navigation menus, forms, and interactive links.

CSS (Styling):

- Style the page with consistent colors, fonts, and spacing.
- Make the website responsive using media queries to adapt to different screen sizes.
- Design clear, intuitive layouts using Flexbox or Grid.

JavaScript (Interactivity):

- Add interactivity such as form validation and user notifications.
- Implement dynamic content updates (e.g., showing or hiding elements).
- Ensure smooth transitions and interactive elements like buttons or modals.

9.3 Using Excel for Data Analysis in Project Planning

Learn how to use Excel to organize and analyze project data, like user details, survey results, or website traffic.

Key Steps:

1. Enter data into Excel (e.g., survey answers or user behavior).
2. Calculate simple stats like averages, most common values, or totals.
3. Make charts or graphs to easily see patterns and trends that help guide your project decisions.

Example of fetching data from webpage as CSV file:

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकुद मन्त्रालय

```
<table id="dataTable">
  <tr><th>Name</th><th>Age</th><th>Country</th></tr>
  <tr><td>John Doe</td><td>30</td><td>USA</td></tr>
  <tr><td>Jane Smith</td><td>25</td><td>UK</td></tr>
</table>
<button onclick="downloadCSV()">Download CSV</button>
<script>
function downloadCSV() {
  let csv = "", rows = document.querySelectorAll("#dataTable tr");
  rows.forEach(row => {
    let cols = row.querySelectorAll("th, td");
    csv += Array.from(cols).map(col => col.textContent).join(",") + "\n";
  });
  let blob = new Blob([csv], { type: "text/csv" });
  let link = document.createElement("a");
  link.href = URL.createObjectURL(blob);
  link.download = "table.csv";
  link.click();
}
</script>
```

सचिव

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकूद मन्त्रालय

9.4 Hands-On Security Testing
Introduction to tools like OWASP ZAP

OWASP ZAP (Zed Attack Proxy) is an open-source web application security testing tool designed to help security professionals identify vulnerabilities in web applications during development and testing. It is part of the OWASP (Open Web Application Security Project) initiative, which aims to improve the security of software and web applications globally. OWASP ZAP is a powerful, flexible tool for testing the security of web applications.

Key features of OWASP ZAP:

- **Automated Scanning:** Detects common web vulnerabilities like SQL injection, XSS, etc.
- **Manual Testing Support:** Tools for in-depth manual testing and exploration.
- **Interception Proxy:** Allows interception and modification of HTTP/HTTPS traffic.
- **Active & Passive Scanning:** Active exploits vulnerabilities and passive scans traffic for issues.
- **Comprehensive Reporting:** Generates detailed vulnerability reports with risk levels and fixes.
- **Extensibility:** Supports plugins and scripting for customization.
- **CI/CD Integration:** Works well in Continuous Integration/Continuous Deployment pipelines.
- **Free & Open Source:** Completely free and open for modification.
- **Cross-Platform:** Available for Windows, Linux, and macOS.
- **User-Friendly:** Offers both GUI and CLI for diverse user needs.

How to use OWASP ZAP (step-by-step Guide)

Step 1: Download and Install OWASP ZAP

1. **Download ZAP**: Go to the official OWASP ZAP website - https://www.zaproxy.org/ and download the appropriate version for your operating system (Windows, Linux, or macOS).

सचिव

2. **Install ZAP**: Follow the installation instructions for your operating system. On Windows, you can use the installer, while on Linux and macOS, you can download the archive and extract it.

Step 2: Launch OWASP ZAP

- After installation, launch OWASP ZAP.
- You will be presented with the main interface, which includes a toolbar, the "Sites" panel (for showing the application structure), and several other panels for different tools (e.g., Active Scan, History, Alerts).

Step 3: Set Up Proxy in Your Browser

OWASP ZAP functions as an intercepting proxy between your browser and the web application you want to test. To use it, you need to configure your browser to route traffic through ZAP.

Start the Proxy:

- ZAP automatically starts its proxy on localhost:8080. This is the address your browser needs to connect to.

Configure Browser:

- Open your browser's network or proxy settings.
- Set the proxy address to localhost and the port to 8080.

- After configuring the proxy, all HTTP(S) traffic will go through ZAP.

Step 4: Explore the Target Application

Explore the Site:

- Open your browser and visit the web application you want to test.
- As you interact with the site (clicking links, submitting forms, etc.), ZAP will capture all the requests and responses in the Sites tab.

View Captured Requests:

- In ZAP, the Sites panel will show the structure of the web application.
- The History tab will list all requests and responses made by your browser during the session.

Step 5: Passive Scanning

1. **Automatic Passive Scanning**:
   - While you browse the site, ZAP automatically performs passive scanning, which involves analyzing the traffic for potential vulnerabilities without actively attacking the site.
   - You can see the results of this scan in the Alerts tab. Alerts will indicate vulnerabilities like missing HTTP headers, insecure cookies, etc.

Step 6: Active Scanning

Start Active Scan:

- After passively exploring the application, you can perform a more aggressive active scan.
- Right-click on the URL or endpoint you want to scan (in the Sites tab).
- Choose Attack > Active Scan.
- Configure the scan settings (e.g., scan policy) and click Start Scan.

Monitor Active Scan:

- The active scan attempts to find vulnerabilities by sending different payloads and analyzing the responses.
- You can monitor the scan progress in the Active Scan tab. The scan will show vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), etc.

Step 7: Analyze Alerts

View Alerts: In the Alerts tab, you can view a list of security issues identified during the scan. Alerts provide information on the severity of each issue, the risk it poses, and the recommended fix.

Example:

- XSS Vulnerability: If an input field is not properly sanitized, an XSS vulnerability may be reported.
- The alert will show where the vulnerability was found, the risk level, and mitigation steps.

Filter Alerts:You can filter alerts by severity, type, or risk level to focus on critical issues.

Step 8: Generate a Report

1. **Generate HTML Report**:
   - Once testing is complete, you can generate a comprehensive report summarizing the results.
   - Go to Report > Generate HTML Report.
   - Choose the location to save the report and select options like including detailed vulnerability descriptions, remediation steps, etc.
   - ZAP will generate an HTML report that you can share with developers or stakeholders.

Step 9: Use ZAP in CI/CD Pipeline (Optional)

OWASP ZAP can also be integrated into Continuous Integration/Continuous Deployment (CI/CD) pipelines to perform security testing automatically.

1. **Install ZAP in CI**:

You can run ZAP via the ZAP Docker Image or use the ZAP API to integrate into Jenkins, GitLab, or other CI tools.

Example using ZAP Docker:

docker run -t owasp/zap2docker-stable zap-baseline.py -t http://yourwebapp.com

2. **Automate Scans**:

   Set up automated scans to run on each build or deployment, so any security issues are identified early in the development cycle.

Example of an Active Scan:

1. Open the **Sites** panel and right-click on a URL (e.g., http://example.com/login).
2. Select **Attack > Active Scan**.
3. In the **Active Scan** window, select the attack types to perform (e.g., SQL injection, XSS).
4. Start the scan and monitor the results in the **Active Scan** tab.

9.5 Test for vulnerabilities: XSS, SQL Injection, insecure cookies

XSS (Cross-Site Scripting) Testing:

- Input Fields: Test all user input fields (e.g., search bars, forms) by entering malicious scripts like <script>alert('XSS')</script>.
- Output Reflection: Check if the input is reflected back in the response without proper sanitization or encoding.
- DOM-based XSS: Look for JavaScript code that manipulates DOM and check if unsanitized input is being executed.

SQL Injection Testing:

- **Input Fields**: Enter SQL-specific characters such as ' OR 1=1 -- in text fields, search boxes, or login forms.
- **Error Messages**: Check if the application shows detailed error messages (e.g., SQL syntax errors), which could indicate vulnerability.
- **Blind SQL Injection**: Test by submitting inputs like 1' AND 1=1 and analyzing response behaviors (e.g., delayed response means a potential vulnerability).

Insecure Cookies Testing:

- **Cookie Attributes**: Check if cookies have Secure and HttpOnly flags enabled, ensuring they are sent only over HTTPS and not accessible via JavaScript.
- **Cookie Manipulation**: Try modifying or deleting cookies using browser developer tools to see if the application can be bypassed or manipulated.

General Tips:

- Use security tools like **OWASP ZAP** or **Burp Suite** for automated vulnerability scanning.
- Always test in a controlled environment (not on live production sites).

माननीय विन्दु कुमार थापा
मन्त्री
सामाजिक विकास, युवा तथा खेलकूद मन्त्रालय

Chapter 10

Final Project Deployment and Showcase

Chapter Summary:

This chapter focuses on preparing, deploying, and showcasing the final project by reviewing, fixing issues, and presenting the completed work effectively.

Learning Objectives:

- Learn to review and debug the project.
- Understand the deployment process for web applications.
- Gain skills to present and explain the project professionally.
- Learn to work in teamwork

10.1 Final Deployment

i. Review the Project:

- Test all features thoroughly to ensure functionality and performance.
- Check for broken links, incorrect paths, or missing assets (e.g., images, CSS, JavaScript).
- Validate HTML, CSS, and JavaScript using online validators to ensure clean code.

ii. Fix Bugs and Issues:

- Use browser developer tools to identify and resolve errors (e.g., console errors or layout issues).
- Optimize the website for performance by compressing images and minimizing CSS/JavaScript.
- Test responsiveness across devices and browsers.

10.2 Presenting the Project

When presenting your project, keep it clear and easy to follow. Focus on showing the main features, explaining how you built it, and sharing any challenges you solved. Keep it simple and interactive to engage the audience.

- Show the main features of the project with screenshots or a live demo.
- Explain how the project was planned, built, and tested.
- Talk about problems you faced and how you solved them.
- Keep it short and easy to understand, and answer questions from the audience.
- Project Presentation must be in group.