HTML

1. **HTML Definition and Purpose:**
HTML (HyperText Markup Language) is the standard language used to create and design web pages on the Internet. Its purpose is to structure content by using a system of tags, which define different elements of a webpage.

Example:
```html
<html>
<head>
    <title>My First HTML Page</title>
</head>
<body>
    <h1>Hello, World!</h1>
    <p>This is a paragraph of text.</p>
</body>
</html>
```

2. **HTML Tags and Attributes:**
HTML tags are used to mark up elements within a document, and attributes provide additional information about those elements or control their behavior.

Example:
```html
<a href="https://www.example.com" target="_blank">Visit Example</a>
<img src="image.jpg" alt="Description of the image">
```

3. **Block-Level vs Inline Elements:**
Block-level elements start on a new line and take up the full width available, while inline elements do not start on a new line and only take up as much width as necessary.

Example:
```html
<div>This is a block-level element.</div>
<span>This is an inline element.</span>
```

4. **Semantic HTML Elements:**
Semantic HTML elements convey meaning rather than just presentation.

Example:
```html
<header>
    <nav>
        <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">About</a></li>
            <li><a href="#">Contact</a></li>
        </ul>
    </nav>
</header>
```

5. **Document Object Model (DOM):**
The DOM represents the structure of a document as a tree of objects, which can be manipulated with scripting languages like JavaScript.

Example:
```html
<script>
document.getElementById("demo").innerHTML = "Hello, JavaScript!";
</script>
<div id="demo"></div>
```

6. **HTML5 vs HTML4 Features:**
HTML5 introduced new features such as semantic elements, multimedia support (`<audio>`, `<video>`), canvas for graphics, and enhanced form controls.

Example:
```html
<video controls>
    <source src="movie.mp4" type="video/mp4">
    Your browser does not support the video tag.
</video>
```

7. **Creating Hyperlinks (`<a>` tag):**
The `<a>` tag creates hyperlinks to other web pages, files, locations within the same page, or email addresses.

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

8. **HTML Forms and Form Elements (`<input>`, `<textarea>`, `<button>`):**
HTML forms allow users to input data. Form elements like `<input>`, `<textarea>`, and `<button>` are used to create interactive fields and buttons.

Example:
```html
<form action="/submit-form" method="post">
    <input type="text" name="username" placeholder="Enter your username"><br>
    <textarea name="message" rows="4" cols="50">Enter your message here...</textarea><br>
    <button type="submit">Submit</button>
</form>
```

9. **Embedding Multimedia Elements (`<audio>`, `<video>`, `<embed>`):**
Multimedia elements like `<audio>`, `<video>`, and `<embed>` are used to embed audio, video, and other media types into web pages.

Example:
```html
<audio controls>
    <source src="audio.mp3" type="audio/mp3">
    Your browser does not support the audio tag.
</audio>
```

10. **Data Attributes (`data-*` attributes):**
`data-*` attributes allow developers to store custom data directly in HTML elements, accessible via JavaScript.

Example:
```html
<div id="product" data-product-id="12345" data-category="electronics">
    Product details here
</div>
```

11. **Creating Tables (`<table>`, `<tr>`, `<td>`, `<th>`):**
Tables in HTML are created using `<table>` for the table itself, `<tr>` for rows, `<td>` for regular cells, and `<th>` for header cells.

Example:
```html
<table border="1">
    <tr>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <td>John</td>
        <td>25</td>
    </tr>
</table>
```

12. **`<meta>` Tag and Its Purpose:**
   The `<meta>` tag provides metadata about the HTML document, such as character encoding, viewport settings, keywords, and descriptions.

   Example:
   ```html
   <meta charset="UTF-8">
   <meta name="description" content="This is a description of the webpage">
   ```

13. **`<div>` vs `<span>`:**
   `<div>` is a block-level element used to group larger sections of content, while `<span>` is an inline element used for smaller chunks of text or inline styling.

   Example:
   ```html
   <div style="color: blue;">This is a block-level element.</div>
   <span style="font-weight: bold;">This is an inline element.</span>
   ```

14. **`<iframe>` Tag Usage:**
   `<iframe>` is used to embed another HTML page within the current page, typically used for embedding maps, videos, or other external content.

   Example:
   ```html
   <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!...">
       Your browser does not support iframes.
   </iframe>
   ```

15. **Including External Resources (CSS, JS):**
   External CSS and JavaScript files are included in HTML documents using `<link>` and `<script>` tags respectively.

   Example:
   ```html
   <link rel="stylesheet" href="styles.css">
   <script src="script.js"></script>
   ```

16. **alt Attribute in Images:**
   The `alt` attribute provides alternative text for images, useful for accessibility and when the image cannot be displayed.

   Example:
   ```html
   <img src="image.jpg" alt="Description of the image">
   ```

17. **Creating Lists (`<ul>`, `<ol>`, `<dl>`):**
   Lists are created using `<ul>` for unordered lists, `<ol>` for ordered lists, and `<dl>` for definition lists.

   Example:
   ```html
   <ul>
       <li>Item 1</li>
       <li>Item 2</li>
   </ul>
   ```

18. **New HTML5 Form Elements (`<datalist>`, `<keygen>`, `<output>`):**
   HTML5 introduced new form elements such as `<datalist>` for providing autocomplete options, `<keygen>` for generating key pairs, and `<output>` for displaying calculation results.

   Example:
   ```html
   <form>
       <input list="browsers">
       <datalist id="browsers">
           <option value="Chrome">
           <option value="Firefox">
       </datalist>
   </form>
   ```

19. **`<canvas>` Element for Graphics:**
   `<canvas>` allows for dynamic rendering of graphics, animations, and interactive content using JavaScript.

   Example:
   ```html
   <canvas id="myCanvas" width="200" height="100"></canvas>
   ```

20. **`<script>`, `<noscript>`, and `<template>` Tags:**
   `<script>` is used to embed or reference JavaScript code, `<noscript>` provides alternate content when JavaScript is disabled, and `<template>` holds client-side content that should not be rendered when the page is loaded but can be instantiated later.

   Example:
   ```html
   <script>
document.getElementById("demo").innerHTML = "Hello, JavaScript!";
   </script>
   <div id="demo"></div>
   ```

21. **Global Attributes (`id`, `class`, `style`, `title`):**
   Global attributes can be applied to any HTML element to provide additional information or to style elements.

   Example:
   ```html
   <div id="main-content" class="container" style="background-color: #f0f0f0;" title="Main Content">
       Content goes here
   </div>
   ```

22. **`<section>` vs `<div>`:**
   `<section>` is a semantic HTML5 element used to define sections in a document, while `<div>` is a generic container.

   Example:
   ```html
   <section>
       <h2>Section Title</h2>
       <p>Section content...</p>
   </section>
   ```

23. **Responsive Layouts with HTML (meta viewport):**
   Responsive layouts are achieved using CSS media queries along with the `<meta>` viewport tag to control the layout on different devices.

   Example:
   ```html
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   ```

24. **`<!DOCTYPE html>` Declaration:**
   Specifies the HTML version and ensures proper rendering by modern web browsers.

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

```
    <link rel="stylesheet" href="styles.css">
    <script src="script.js" defer></script>
</head>
<body>
    <header>
        <h1>Website Header</h1>
        <nav>
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
        </nav>
    </header>

    <section>
        <h2>Main Content Section</h2>
        <p>This section contains the main content of the webpage.</p>
    </section>

    <footer>
        <p>&copy; 2024 My Website. All rights reserved.</p>
    </footer>
</body>
</html>
```

25. **Handling Broken Images (`onerror` attribute):**
   The `onerror` attribute in `<img>` tags allows specifying a JavaScript function to handle situations where an image fails to load.

   Example:
   ```html
   <img src="image.jpg" alt="Description of image" onerror="this.src='placeholder.jpg';">
   ```

These examples illustrate how HTML elements, attributes, and features are used to create structured, semantic, and interactive web pages.

CSS

Certainly! Here are explanations and examples for each CSS topic:

1. **CSS Definition and Purpose:**
  CSS (Cascading Style Sheets) is a language used for describing the presentation of a document written in HTML or XML. It controls the layout, colors, fonts, and other visual aspects of web pages.

   Example:
   ```html
   <style>
       body {
           font-family: Arial, sans-serif;
           background-color: #f0f0f0;
       }
       h1 {
           color: blue;
           text-align: center;
       }
   </style>
   ```

2. **CSS Selectors:**
   CSS selectors are patterns used to select and style elements in an HTML document.

   Example:
   ```css
   /* Element selector */
   p {
       font-size: 16px;
   }

   /* Class selector */
   .important {
       color: red;
   }

   /* ID selector */
   #header {
       background-color: #333;
       color: white;
   }
   ```

3. **CSS Specificity and Inheritance:**
   Specificity determines which CSS rule applies to an element. Inheritance allows styles applied to parent elements to be inherited by their child elements.

   Example:
   ```html
   <style>
       /* Specificity example */
       .container div {
           color: blue;  /* Lower specificity */
       }
       div#special {
           color: red;  /* Higher specificity */
       }

       /* Inheritance example */
       .parent {
           font-size: 18px;
       }
       .child {
           /* Inherits font-size from parent */
       }
   </style>
   ```

4. **CSS Box Model:**
   The CSS box model describes the rectangular boxes that are generated for elements in the document tree.

   Example:
   ```css
   .box {
       width: 200px;
       padding: 20px;
       border: 1px solid #ccc;
       margin: 10px;
   }
   ```

5. **Differences Between Classes and IDs:**
   Classes can be used multiple times in an HTML document, while IDs should be unique. IDs have higher specificity than classes.

   Example:
   ```html
   <div class="box">This is a box.</div>
   <div id="unique-box">This is a unique box.</div>
   ```

6. **Applying CSS (Inline, Internal, External):**
   CSS can be applied inline within HTML tags, internally in `<style>` tags within the `<head>` of an HTML document, or externally via separate CSS files linked to the HTML document.

   Example:
   ```html
   <!-- Inline CSS -->
   <div style="color: red;">This text is red.</div>

   <!-- Internal CSS -->
   <style>
       .paragraph {
           font-size: 16px;
           line-height: 1.5;
       }
   </style>
   <div class="paragraph">This is a paragraph styled internally.</div>

   <!-- External CSS -->
   <link rel="stylesheet" href="styles.css">
   ```

7. **CSS Positioning (Static, Relative, Absolute, Fixed, Sticky):**
   CSS positioning determines how an element is positioned in the document flow.

   Example:
   ```css
   .box {
   ```

```
    position: relative; /* or absolute, fixed, static, sticky */
    top: 20px;
    left: 50px;
  }
```

8. **CSS Flexbox Layout:**
  Flexbox is a layout model that allows responsive elements within a container to be automatically arranged depending on the screen size.

  Example:
  ```css
  .container {
     display: flex;
     justify-content: space-around;
     align-items: center;
  }
  ```

9. **CSS Grid Layout:**
  CSS Grid Layout is a two-dimensional layout system for CSS that allows you to design complex web layouts more easily.

  Example:
  ```css
  .grid-container {
     display: grid;
     grid-template-columns: 1fr 1fr 1fr;
     grid-gap: 10px;
  }
  ```

10. **CSS Display Property:**
   The `display` property specifies the display behavior of an element.

   Example:
   ```css
   .inline {
      display: inline;
   }
   .block {
      display: block;
   }
   ```

11. **CSS Pseudo-classes and Pseudo-elements:**
   Pseudo-classes and pseudo-elements style elements based on their state or position in the document tree.

   Example:
   ```css
   /* Pseudo-class */
   a:hover {
      color: red;
   }

   /* Pseudo-element */
   p::first-line {
      font-weight: bold;
   }
   ```

12. **CSS Transitions and Animations:**
   CSS transitions and animations allow you to animate changes to CSS properties.

   Example:
   ```css
   .box {
      width: 100px;
      height: 100px;
      background-color: blue;
      transition: width 2s, height 2s;
   }
   .box:hover {
      width: 200px;
      height: 200px;
   }
   ```

13. **CSS Media Queries for Responsive Design:**
   Media queries allow styles to be applied depending on the characteristics of the device, such as screen width, height, and orientation.

   Example:
   ```css
   @media screen and (max-width: 600px) {
      body {
         background-color: lightblue;
      }
   }
   ```

14. **Differences Between `display: none` and `visibility: hidden`:**
   `display: none` removes the element from the document flow, whereas `visibility: hidden` hides the element but keeps its space in the layout.

   Example:
   ```css
   .hidden {
      display: none;
   }
   .invisible {
      visibility: hidden;
   }
   ```

15. **CSS `float` Property and Clearfix:**
   The `float` property positions an element to the left or right in its container. Clearfix is used to clear floated elements.

   Example:
   ```css
   .float-left {
      float: left;
   }
   .clearfix::after {
      content: "";
      display: table;
      clear: both;
   }
   ```

16. **CSS Z-index and Stacking Context:**
   `z-index` specifies the stacking order of positioned elements. Stacking context defines how elements are stacked relative to each other.

   Example:
   ```css
   .box1 {
      position: relative;
      z-index: 2;
   }
   .box2 {
      position: relative;
      z-index: 1;
   }
   ```

17. **CSS Units (px, em, rem, %, vw, vh):**
   CSS units determine the size or position of elements relative to other elements or the viewport.

   Example:
   ```css
   .box {
      width: 200px;
      font-size: 1.2em;
      padding: 2rem;
      margin-top: 10%;
      height: 50vw;
   }
   ```

18. **CSS Variables (Custom Properties):**
   CSS variables allow for reusable values throughout a stylesheet.

   Example:
   ```css
   :root {
      --primary-color: #007bff;
   }
   .box {
      color: var(--primary-color);
   }
   ```

19. **CSS `box-sizing` Property:**
   The `box-sizing` property defines how the total width and height of an element is calculated.

   Example:
   ```css
   .box {
      box-sizing: border-box;
      width: 200px;
      padding: 20px;
   ```

    border: 1px solid black;
}

20. **CSS Combinators (Descendant, Child, Adjacent Sibling, General Sibling):**
CSS combinators are used to select elements based on their relationship with other elements.

Example:
```css
/* Descendant combinator */
.container p {
    font-size: 16px;
}
/* Child combinator */
.container > ul {
    list-style-type: none;
}
/* Adjacent sibling combinator */
h1 + p {
    margin-top: 0;
}
/* General sibling combinator */
h2 ~ p {
    font-style: italic;
}
```

21. **CSS Sprites:**
CSS sprites combine multiple images into a single image, reducing the number of server requests.

Example:
```css
.icon {
    background-image:
url('sprites.png');
    background-position: -20px
-40px;
    width: 30px;
    height: 30px;
}
```

22. **Centering Elements Horizontally and Vertically:**
CSS techniques to center elements horizontally and vertically within their containers.

Example:
```css
.center {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%,
-50%);
}
```

23. **CSS Preprocessors (SASS, LESS):**
CSS preprocessors extend CSS with variables, nested rules, mixins, and functions, which are then compiled into standard CSS.

Example (SASS):
```scss
$primary-color: #007bff;
.box {
    color: $primary-color;
}
```

24. **Including Custom Fonts (`@font-face`, Google Fonts):**
Custom fonts can be included using `@font-face` for locally hosted fonts or Google Fonts for web fonts.

Example (`@font-face`):
```css
@font-face {
    font-family: 'MyCustomFont';
    src: url('mycustomfont.woff2')
format('woff2'),
         url('mycustomfont.woff')
format('woff');
    font-weight: normal;
    font-style: normal;
}
.custom-font {
    font-family: 'MyCustomFont',
Arial, sans-serif;
}
```

Example (Google Fonts):
```html
<link rel="stylesheet"
href="https://fonts.googleapis.com/cs
s?family=Roboto">
<style>
    .google-font {
        font-family: 'Roboto',
sans-serif;
    }
</style>
```

25. **Handling Browser Compatibility (Vendor Prefixes, Feature Queries, Polyfills):**
Techniques to ensure CSS features work across different browsers, including using vendor prefixes, feature queries, and polyfills.

Example (Vendor Prefix):
```css
.box {
    display: -webkit-box;  /* Safari,
iOS, Android */
    display: -moz-box;     /* Firefox
*/
    display: box;          /* Standard
*/
}
```

Example (Feature Query):
```css
@supports (display: grid) {
    .grid-container {
        display: grid;
        grid-template-columns: 1fr
1fr;
    }
}
```

Example (Polyfill):
```html
<script
src="https://cdn.polyfill.io/v3/polyfill.
min.js"></script>
```

These examples cover various aspects of CSS, from basic styling to advanced layout techniques and ensuring compatibility across different browsers.

Front End

Given the extensive list of topics, I'll answer each with concise explanations and examples where applicable:

1. **HTML Definition and Purpose:**
HTML (HyperText Markup Language) is the standard language used to create and design web pages on the Internet. Its purpose is to structure content using a system of tags that define different elements of a webpage.

Example:
```html
<!DOCTYPE html>
<html>
<head>
    <title>My First HTML
Page</title>
</head>
<body>
    <h1>Hello, World!</h1>
    <p>This is a paragraph of
text.</p>
</body>
</html>
```

2. **CSS Selectors:**
CSS selectors are patterns used to select and style elements in HTML documents.

Example:
```css
/* Selects all <p> elements */
p {
    color: blue;
}
```

```
  /* Selects elements with class "container" */
  .container {
     width: 100%;
  }

  /* Selects element with id "header" */
  #header {
     background-color: #ccc;
  }
  ```

3. **JavaScript Functions and Scope:**
  Functions in JavaScript are blocks of reusable code. Scope defines where variables and functions are accessible within a program.

  Example:
  ```javascript
  // Function declaration
  function greet(name) {
     return `Hello, ${name}!`;
  }

  // Function call
  console.log(greet('Alice')); // Outputs: Hello, Alice!

  // Scope example
  let x = 10;
  function testScope() {
     let y = 20;
     console.log(x); // Accessible: Outputs 10
     console.log(y); // Accessible: Outputs 20
  }
  console.log(x); // Accessible: Outputs 10
  console.log(y); // Not accessible: Throws ReferenceError
  ```

4. **Block-Level vs Inline Elements:**
  Block-level elements start on a new line and take up the full width available, while inline elements do not start on a new line and only take up as much width as necessary.

  Example:
  ```html
  <div>This is a block-level element.</div>
  <span>This is an inline element.</span>
  ```

5. **CSS Specificity and Inheritance:**
  Specificity determines which CSS rule is applied when multiple rules could apply to the same element. Inheritance allows styles applied to parent elements to be inherited by their child elements.

  Example:
  ```html
  <style>
     /* Specificity example */
     p {
        color: blue;  /* Specificity: 0, 0, 1 */
     }
     .special {
        color: red;   /* Specificity: 0, 0, 10 */
     }
     #unique {
        color: green; /* Specificity: 0, 1, 0 */
     }
  </style>
  <p>This text is blue.</p>
  <p class="special">This text is red.</p>
  <p id="unique">This text is green.</p>
  ```

6. **Data Types and Variables:**
  JavaScript has several data types including strings, numbers, booleans, objects, arrays, and more. Variables are used to store data values.

  Example:
  ```javascript
  let message = 'Hello';        // String
  let number = 42;              // Number
  let isTrue = true;            // Boolean
  let person = {                // Object
     name: 'Alice',
     age: 30
  };
  let colors = ['red', 'blue']; // Array
  ```

7. **Semantic HTML Elements:**
  Semantic HTML elements convey meaning rather than just presentation.

  Example:
  ```html
  <header>
     <nav>
        <ul>
           <li><a href="#">Home</a></li>
           <li><a href="#">About</a></li>
           <li><a href="#">Contact</a></li>
        </ul>
     </nav>
  </header>
  ```

8. **CSS Box Model:**
  The CSS box model describes the rectangular boxes that are generated for elements in the document tree. It consists of margins, borders, padding, and the actual content area of an element.

  Example:
  ```html
  <style>
     .box {
        width: 200px;
        height: 100px;
        padding: 20px;
        border: 1px solid #000;
        margin: 10px;
        background-color: #ccc;
     }
  </style>
  <div class="box">Content</div>
  ```

9. **ES6 Features (let, const, arrow functions, template literals, etc.):**
  ES6 introduced several new features to JavaScript such as `let` and `const` for variable declaration, arrow functions for concise function syntax, and template literals for easier string interpolation.

  Example:
  ```javascript
  // Arrow function example
  const add = (a, b) => a + b;

  // Template literal example
  let name = 'Alice';
  console.log(`Hello, ${name}!`);
  ```

10. **Document Object Model (DOM):**
   The DOM is a programming interface for web documents. It represents the structure of a document as a tree of objects, allowing JavaScript to manipulate and modify the content and structure of a webpage.

   Example:
   ```html
   <script>
document.getElementById("demo").innerHTML = "Hello, JavaScript!";
   </script>
   <div id="demo"></div>
   ```

These examples cover a range of fundamental topics in HTML, CSS, and JavaScript, providing clear explanations and practical code snippets where applicable. Let me know if you'd like to continue with more topics!

Certainly! Let's continue with more topics:

11. **Differences Between Classes and IDs:**

Classes and IDs are both used for styling HTML elements, but IDs are unique and can only be applied to one element per page, while classes can be applied to multiple elements.

Example:
```html
<style>
    /* Class selector */
    .highlight {
        background-color: yellow;
    }

    /* ID selector */
    #unique {
        color: blue;
    }
</style>
<p class="highlight">This paragraph has a class.</p>
<p id="unique">This paragraph has an ID.</p>
```

12. **JavaScript Arrays and Objects:**
Arrays and objects are used to store multiple values in JavaScript. Arrays are ordered lists of values, while objects are collections of key-value pairs.

Example:
```javascript
// Array example
let colors = ['red', 'green', 'blue'];

// Object example
let person = {
    name: 'Alice',
    age: 30,
    city: 'New York'
};
```

13. **HTML5 vs HTML4 Features:**
HTML5 introduced new features such as semantic elements (`<header>`, `<footer>`), multimedia support (`<audio>`, `<video>`), canvas for graphics, local storage, and improved form controls.

Example:
```html
<!-- Video element in HTML5 -->
<video controls>
    <source src="movie.mp4" type="video/mp4">
    Your browser does not support the video tag.
</video>
```

14. **Applying CSS (Inline, Internal, External):**
CSS can be applied inline (directly in the HTML), internally (within a `<style>` tag in the `<head>`), or externally (linking to an external CSS file).

Example:
```html
<!-- Inline CSS -->
<div style="color: blue;">This is styled inline.</div>

<!-- Internal CSS -->
<style>
    .container {
        width: 80%;
        margin: 0 auto;
    }
</style>

<!-- External CSS -->
<link rel="stylesheet" href="styles.css">
```

15. **DOM Manipulation with JavaScript:**
DOM manipulation involves modifying the structure, style, or content of a webpage using JavaScript.

Example:
```html
<script>
    // Change text content of an element
    document.getElementById("demo").innerHTML = "Updated content";

    // Create a new element and append it to the DOM
    let newDiv = document.createElement('div');
    newDiv.textContent = 'New Element';
    document.body.appendChild(newDiv);
</script>
<div id="demo">Initial content</div>
```

16. **Creating Hyperlinks (`<a>` tag):**
The `<a>` tag creates hyperlinks to other web pages, files, locations within the same page, or email addresses.

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

17. **CSS Positioning (Static, Relative, Absolute, Fixed, Sticky):**
CSS positioning determines how an element is positioned within its parent container or the viewport.

Example:
```html
<style>
    .box {
        position: relative; /* Relative positioning */
        top: 20px;
        left: 30px;
    }

    .fixed {
        position: fixed; /* Fixed positioning */
        top: 0;
        right: 0;
    }
</style>
<div class="box">Relative Positioning</div>
<div class="fixed">Fixed Positioning</div>
```

18. **Event Handling in JavaScript:**
Event handling allows JavaScript to respond to user actions like clicks, mouse movements, key presses, etc.

Example:
```html
<button onclick="alert('Button clicked!')">Click Me</button>
```

19. **HTML Forms and Form Elements (`<input>`, `<textarea>`, `<button>`):**
HTML forms allow users to input data using form elements like `<input>`, `<textarea>`, `<button>`, `<select>`, etc.

Example:
```html
<form action="/submit-form" method="post">
    <input type="text" name="username" placeholder="Enter your username"><br>
    <textarea name="message" rows="4" cols="50">Enter your message here...</textarea><br>
    <button type="submit">Submit</button>
</form>
```

20. **CSS Flexbox Layout:**
Flexbox is a layout model in CSS designed for laying out elements in a single direction—either horizontally or vertically.

Example:

```html
<style>
    .flex-container {
        display: flex;
        justify-content:
space-between;
    }
    .flex-item {
        flex: 1;
    }
</style>
<div class="flex-container">
    <div class="flex-item">Item
1</div>
    <div class="flex-item">Item
2</div>
    <div class="flex-item">Item
3</div>
</div>
```

These examples provide practical illustrations of various HTML, CSS, and JavaScript concepts. Let me know if you'd like to continue with more topics or if there's something specific you'd like to explore further! Certainly! Let's continue exploring more topics:

21. **JavaScript Promises and Async/Await:**
Promises are used for asynchronous programming in JavaScript, allowing execution of code that depends on the result of a previous operation. Async functions simplify asynchronous code with the `async` and `await` keywords.

Example:
```javascript
// Using Promises
function getData() {
    return new Promise((resolve,
reject) => {
        setTimeout(() => {
            resolve('Data fetched
successfully');
        }, 2000);
    });
}

getData().then(data => {
    console.log(data); // Outputs:
Data fetched successfully
}).catch(error => {
    console.error(error);
});

// Using async/await
async function fetchData() {
    try {
        let data = await getData();
        console.log(data); // Outputs:
Data fetched successfully
    } catch (error) {
        console.error(error);
    }
}

fetchData();
```

22. **Embedding Multimedia Elements (`<audio>`, `<video>`, `<embed>`):**
Multimedia elements like `<audio>`, `<video>`, and `<embed>` are used to embed audio, video, and other media types into web pages.

Example:
```html
<video controls>
    <source src="movie.mp4"
type="video/mp4">
    Your browser does not support
the video tag.
</video>
```

23. **CSS Grid Layout:**
CSS Grid Layout provides a two-dimensional grid-based layout system, making it easier to design complex web layouts.

Example:
```html
<style>
    .grid-container {
        display: grid;
        grid-template-columns: 1fr 1fr
1fr;
        gap: 10px;
    }
    .grid-item {
        background-color: #ccc;
        padding: 20px;
    }
</style>
<div class="grid-container">
    <div class="grid-item">Item
1</div>
    <div class="grid-item">Item
2</div>
    <div class="grid-item">Item
3</div>
</div>
```

24. **JavaScript Closures:**
Closures are functions that remember the lexical scope in which they were created, even when the function is executed outside that scope.

Example:
```javascript
function outerFunction() {
    let message = 'Hello';
    function innerFunction() {
        console.log(message); //
Accesses variable from outer
function's scope
    }
    return innerFunction;
}

let myFunction = outerFunction();
myFunction(); // Outputs: Hello
```

25. **Data Attributes (`data-*` attributes):**
`data-*` attributes allow developers to store custom data directly in HTML elements, accessible via JavaScript.

Example:
```html
<div id="product"
data-product-id="12345"
data-category="electronics">
    Product details here
</div>
<script>
    let productDiv =
document.getElementById('product');

console.log(productDiv.dataset.prod
uctId); // Outputs: 12345

console.log(productDiv.dataset.categ
ory); // Outputs: electronics
</script>
```

26. **CSS Display Property:**
The CSS `display` property specifies the display behavior (block, inline, flex, etc.) of an element.

Example:
```html
<style>
    .block {
        display: block;
        width: 100px;
        height: 100px;
        background-color: red;
    }

    .inline {
        display: inline;
        width: 100px;
        height: 100px;
        background-color: blue;
    }
</style>
<div class="block">Block
Element</div>
<span class="inline">Inline
Element</span>
```

27. **JavaScript Prototypes and Inheritance:**
JavaScript uses prototypes to implement inheritance between objects. Objects inherit properties and methods from their prototype chain.

Example:
```javascript
```

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

Person.prototype.greet = function() {
    return `Hello, my name is ${this.name}.`;
};

function Student(name, age, grade) {
    Person.call(this, name, age);
    this.grade = grade;
}

Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;

let student = new Student('Alice', 18, 'A');
console.log(student.greet()); // Outputs: Hello, my name is Alice.
```

28. **Creating Tables (`<table>`, `<tr>`, `<td>`, `<th>`):**
   Tables in HTML are created using `<table>` for the table itself, `<tr>` for rows, `<td>` for regular cells, and `<th>` for header cells.

   Example:
```html
<table border="1">
    <tr>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <td>John</td>
        <td>25</td>
    </tr>
</table>
```

29. **CSS Pseudo-classes and Pseudo-elements:**
   CSS pseudo-classes and pseudo-elements allow styling based on certain conditions or positions in the document.

   Example:
```html
<style>
    /* Pseudo-class example */
    a:hover {
        color: red;
    }

    /* Pseudo-element example */
    p::first-line {
        font-weight: bold;
    }
</style>
<a href="#">Hover me</a>
<p>This is the first line of text.</p>
```

30. **Error Handling in JavaScript:**
   Error handling in JavaScript involves using `try...catch` blocks to manage exceptions and errors that may occur during execution.

   Example:
```javascript
try {
    // Code that may throw an error
    let result = 1 / 0; // This will throw a division by zero error
} catch (error) {
    // Handle the error
    console.error('An error occurred:', error.message);
}
```

These examples cover a range of intermediate topics in HTML, CSS, and JavaScript, providing clear explanations and practical code snippets where applicable. Let me know if there's something specific you'd like to explore further!
Certainly! Let's continue with more topics:

31. **`<meta>` Tag and Its Purpose:**
   The `<meta>` tag in HTML is used to provide metadata about the HTML document. It includes information such as character encoding, viewport settings for responsive design, keywords, and descriptions for SEO.

   Example:
```html
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="This is a description of the webpage">
```

32. **CSS Transitions and Animations:**
   CSS transitions and animations are used to create visual effects and animations in web pages without relying on JavaScript.

   Example (CSS Transition):
```html
<style>
    .box {
        width: 100px;
        height: 100px;
        background-color: red;
        transition: width 1s ease-in-out;
    }

    .box:hover {
        width: 200px;
    }
</style>
<div class="box"></div>
```

   Example (CSS Animation):
```html
<style>
    @keyframes color-change {
        0% { background-color: red; }
        50% { background-color: yellow; }
        100% { background-color: green; }
    }

    .animated-box {
        width: 100px;
        height: 100px;
        background-color: red;
        animation: color-change 3s infinite;
    }
</style>
<div class="animated-box"></div>
```

33. **JavaScript Modules (import/export):**
   JavaScript modules provide a way to structure and organize code into separate files or modules, allowing components to be imported and exported for use in other parts of the application.

   Example (Module Export):
```javascript
// math.js
export function add(a, b) {
    return a + b;
}

// main.js
import { add } from './math.js';

console.log(add(5, 3)); // Outputs: 8
```

34. **`<div>` vs `<span>`:**
   `<div>` and `<span>` are HTML elements used for grouping content and applying styles.

   Example:
```html
<div style="color: blue;">This is a block-level element.</div>
<span style="font-weight: bold;">This is an inline element.</span>
```

35. **CSS Media Queries for Responsive Design:**

CSS media queries are used to apply different styles based on the device characteristics such as screen width, height, orientation, etc., enabling responsive web design.

Example:
```html
<style>
    /* Default styles */
    .container {
        width: 100%;
        max-width: 1200px;
        margin: 0 auto;
        padding: 20px;
    }

    /* Media query for tablets and smaller screens */
    @media (max-width: 768px) {
        .container {
            padding: 10px;
        }
    }

    /* Media query for mobile phones */
    @media (max-width: 480px) {
        .container {
            padding: 5px;
        }
    }
</style>
<div class="container">
    <!-- Content goes here -->
</div>
```

36. **JavaScript Array Methods (map, filter, reduce, etc.):**
JavaScript provides powerful array methods such as `map`, `filter`, `reduce`, etc., for manipulating arrays and performing operations like transformation, filtering, and aggregation.

Example (Array `map`):
```javascript
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map(x => x * 2);
console.log(doubled); // Outputs: [2, 4, 6, 8, 10]
```

Example (Array `filter`):
```javascript
const scores = [80, 90, 60, 45, 70];
const passedScores = scores.filter(score => score >= 70);
console.log(passedScores); // Outputs: [80, 90, 70]
```

37. **`<iframe>` Tag Usage:**
`<iframe>` is used to embed another HTML page within the current page, typically used for embedding maps, videos, or other external content.

Example:
```html
<iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!...">
    Your browser does not support iframes.
</iframe>
```

38. **Differences Between `display: none` and `visibility: hidden`:**
`display: none` removes an element from the document flow, and it won't take up any space on the page, while `visibility: hidden` hides the element, but the space it occupies remains.

Example:
```html
<style>
    .hidden-element {
        display: none; /* Element is completely removed from the layout */
        visibility: hidden; /* Element is hidden, but its space is preserved */
    }
</style>
<div class="hidden-element">Hidden Content</div>
```

39. **JavaScript Callbacks:**
Callbacks are functions passed as arguments to other functions to be executed later, often used in asynchronous programming or event handling.

Example:
```javascript
function fetchData(callback) {
    setTimeout(() => {
        const data = 'Data fetched successfully';
        callback(data);
    }, 2000);
}

function displayData(data) {
    console.log(data);
}

fetchData(displayData); // Outputs: Data fetched successfully
```

40. **Including External Resources (CSS, JS):**
External CSS and JavaScript files are included in HTML documents using `<link>` and `<script>` tags respectively.

Example:
```html
<link rel="stylesheet" href="styles.css">
<script src="script.js"></script>
```

These examples cover a wide range of topics in HTML, CSS, and JavaScript, providing practical examples and explanations to help understand each concept better. If you have more specific topics or questions, feel free to ask!
Sure, let's continue with more topics:

41. **CSS float Property and Clearfix:**
The `float` property in CSS is used to place an element to the left or right of its container, allowing content to flow around it. When using floats, the clearfix technique is often used to clear floats and prevent layout issues.

Example:
```html
<style>
    .container {
        border: 1px solid #ccc;
    }

    .left {
        float: left;
        width: 200px;
        height: 150px;
        background-color: lightblue;
        margin-right: 10px;
    }

    .right {
        float: right;
        width: 200px;
        height: 150px;
        background-color: lightgreen;
        margin-left: 10px;
    }

    .clearfix::after {
        content: "";
        display: table;
        clear: both;
    }
</style>
<div class="container clearfix">
    <div class="left">Left Content</div>
    <div class="right">Right Content</div>
</div>
```

42. **AJAX and Fetch API:**
AJAX (Asynchronous JavaScript and XML) is a technique used to send and receive data from a server

asynchronously without reloading the entire page. The Fetch API is a modern replacement for XMLHttpRequest (XHR) and provides a more powerful and flexible interface for fetching resources.

   Example (Fetch API):
```javascript

fetch('https://api.example.com/data')
       .then(response =>
response.json())
       .then(data =>
console.log(data))
       .catch(error =>
console.error('Error fetching data:',
error));
```

43. **alt Attribute in Images:**
   The `alt` attribute in HTML is used to provide alternative text for images. It is important for accessibility purposes, helping users with visual impairments understand the content of the image.

   Example:
```html
   <img src="image.jpg"
alt="Description of the image">
```

44. **CSS Z-index and Stacking Context:**
   The `z-index` property in CSS controls the stacking order of positioned elements. Elements with a higher `z-index` value are stacked in front of elements with lower values. Understanding stacking context is crucial when dealing with complex layouts.

   Example:
```html
   <style>
      .back {
         position: absolute;
         background-color: lightblue;
         width: 200px;
         height: 200px;
         z-index: 1;
      }

      .front {
         position: absolute;
         background-color: lightgreen;
         width: 150px;
         height: 150px;
         top: 50px;
         left: 50px;
         z-index: 2;
      }
   </style>
   <div class="back"></div>
   <div class="front"></div>
```

45. **JavaScript Event Loop and Concurrency Model:**
   The JavaScript event loop is responsible for handling asynchronous operations in JavaScript. It follows a single-threaded, non-blocking concurrency model where tasks are queued and executed in a loop.

   Example (Event Loop):
```javascript
   console.log('Start');

   setTimeout(() => {
      console.log('Async Task');
   }, 2000);

   console.log('End');
```

46. **Creating Lists (`<ul>`, `<ol>`, `<dl>`):**
   Lists in HTML can be created using `<ul>` for unordered lists, `<ol>` for ordered lists, and `<dl>` for definition lists.

   Example:
```html
   <ul>
      <li>Item 1</li>
      <li>Item 2</li>
   </ul>

   <ol>
      <li>Item 1</li>
      <li>Item 2</li>
   </ol>

   <dl>
      <dt>Term 1</dt>
      <dd>Definition 1</dd>
      <dt>Term 2</dt>
      <dd>Definition 2</dd>
   </dl>
```

47. **CSS Units (px, em, rem, %, vw, vh):**
   CSS units define the size of elements. `px` is pixel, `em` is relative to the font-size of the element, `rem` is relative to the font-size of the root element, `%` is relative to the parent element, `vw` is relative to 1% of the viewport width, and `vh` is relative to 1% of the viewport height.

   Example:
```html
   <style>
      .box {
         width: 200px; /* Fixed width
in pixels */
         font-size: 1.2em; /* Relative
font size */
         padding: 2rem; /* Relative
padding based on root font-size */
         margin-top: 10%; /* Margin
relative to parent's width */
         height: 50vw; /* Height
relative to viewport width */
      }
   </style>
   <div class="box">Box with
different CSS units</div>
```

48. **JavaScript Local Storage and Session Storage:**
   Local Storage and Session Storage are mechanisms in JavaScript used to store key-value pairs locally in a user's browser. Local Storage persists until explicitly deleted, while Session Storage persists only for the duration of the page session.

   Example (Local Storage):
```javascript
   localStorage.setItem('username',
'John');
   const username =
localStorage.getItem('username');
   console.log(username); //
Outputs: John
```

49. **New HTML5 Form Elements (`<datalist>`, `<keygen>`, `<output>`):**
   HTML5 introduced new form elements such as `<datalist>` for providing autocomplete options, `<keygen>` for generating key pairs, and `<output>` for displaying calculation results.

   Example (`<datalist>`):
```html
   <form>
      <input list="browsers">
      <datalist id="browsers">
         <option value="Chrome">
         <option value="Firefox">
      </datalist>
   </form>
```

50. **CSS Variables (Custom Properties):**
   CSS Variables (Custom Properties) allow you to define variables directly in CSS, which can be reused throughout the stylesheet.

   Example:
```html
   <style>
      :root {
         --main-color: #3498db;
         --padding: 10px;
      }

      .box {
         background-color: var(--main-color);
         padding: var(--padding);
      }
   </style>
   <div class="box">Box with CSS Variables</div>
```

These examples cover a broad range of topics in HTML, CSS, and JavaScript, illustrating practical applications and concepts. If you have more specific questions or need further clarification on any topic, feel free to ask!
Certainly! Let's continue with more topics:

51. **JSON Parsing and Stringification:**
   JSON (JavaScript Object Notation) is a lightweight data interchange format. Parsing JSON converts a JSON string into a JavaScript object, while stringification converts a JavaScript object into a JSON string.

   Example (Parsing JSON):
```javascript
   const jsonString = '{"name": "John", "age": 30}';
   const obj = JSON.parse(jsonString);
   console.log(obj.name); // Outputs: John
```

52. **`<canvas>` Element for Graphics:**
   The `<canvas>` element in HTML5 is used to draw graphics, animations, and other visual content dynamically using JavaScript.

   Example:
```html
   <canvas id="myCanvas" width="200" height="100"></canvas>
   <script>
      const canvas = document.getElementById('myCanvas');
      const ctx = canvas.getContext('2d');
      ctx.fillStyle = 'red';
      ctx.fillRect(10, 10, 50, 50);
   </script>
```

53. **CSS `box-sizing` Property:**
   The `box-sizing` property in CSS controls how the total width and height of an element is calculated. The default value is `content-box`, but `border-box` includes padding and border in the element's total width and height.

   Example:
```html
   <style>
      .box {
         width: 200px;
         height: 100px;
         padding: 20px;
         border: 1px solid black;
         box-sizing: border-box; /* Include padding and border in total width */
      }
   </style>
   <div class="box">Box with box-sizing: border-box</div>
```

54. **JavaScript Regular Expressions:**
   Regular expressions (RegEx) in JavaScript are used for matching patterns in strings. They provide a powerful way to search, replace, and extract information from text based on patterns.

   Example:
```javascript
   const regex = /\d+/; // Matches one or more digits
   const text = 'Hello, 123 world!';
   const result = text.match(regex);
   console.log(result); // Outputs: ["123"]
```

55. **`<script>`, `<noscript>`, and `<template>` Tags:**
   - `<script>` is used to embed or reference JavaScript code in HTML.
   - `<noscript>` provides alternate content when JavaScript is disabled.
   - `<template>` holds client-side content that should not be rendered when the page is loaded but can be instantiated later.

   Example (`<template>`):
```html
   <template id="myTemplate">
      <h2>Title</h2>
      <p>Content goes here...</p>
   </template>
   <script>
      const template = document.getElementById('myTemplate');
      const clone = document.importNode(template.content, true);

document.body.appendChild(clone);
   </script>
```

56. **CSS Combinators (Descendant, Child, Adjacent Sibling, General Sibling):**
   CSS combinators are used to select elements based on their relationship with other elements.
   - Descendant combinator (` `) selects an element that is a descendant of another element.
   - Child combinator (`>`) selects an element that is a direct child of another element.
   - Adjacent sibling combinator (`+`) selects an element that is directly adjacent (immediately follows) another element.
   - General sibling combinator (`~`) selects an element that is a sibling to another element.

   Example:
```html
   <style>
      /* Descendant selector */
      div p {
         color: blue;
      }

      /* Child selector */
      div > p {
         font-weight: bold;
      }

      /* Adjacent sibling selector */
      h1 + p {
         margin-top: 0;
      }

      /* General sibling selector */
      h2 ~ p {
         font-style: italic;
      }
   </style>
   <div>
      <p>Paragraph inside a div.</p>
      <h1>Heading 1</h1>
      <p>Adjacent paragraph.</p>
      <h2>Heading 2</h2>
      <p>General sibling paragraph.</p>
   </div>
```

57. **JavaScript Hoisting:**
   Hoisting in JavaScript refers to the behavior where variable and function declarations are moved to the top of their containing scope during compilation, regardless of where they are declared.

   Example:
```javascript
   console.log(myVar); // Outputs: undefined
   var myVar = 'Hello';

   // After hoisting by the interpreter
   var myVar;
   console.log(myVar); // Outputs: undefined
   myVar = 'Hello';
```

58. **Global Attributes (`id`, `class`, `style`, `title`):**
   Global attributes can be applied to any HTML element and include:
   - `id`: Specifies a unique identifier for an element.
   - `class`: Specifies one or more class names for an element.
   - `style`: Specifies inline CSS styles for an element.
   - `title`: Specifies extra information about an element (tooltip text).

   Example:
   ```html
   <div id="myDiv" class="container"
style="background-color: lightblue;"
title="This is a container">
      Content goes here...
   </div>
   ```

59. **CSS Sprites:**
   CSS sprites combine multiple images into a single image file, reducing the number of HTTP requests made by a browser for images. Sprites are then used with CSS background positioning to display specific images.

   Example:
   ```css
   .icon {
      width: 32px;
      height: 32px;
      background-image:
url('sprites.png');
   }

   .icon-home {
      background-position: 0 0;
   }

   .icon-email {
      background-position: -32px 0;
   }
   ```

60. **JavaScript `this` Keyword:**
   The `this` keyword in JavaScript refers to the object to which a function belongs, or the object that is executing the current function. Its value is determined by how a function is called.

   Example:
   ```javascript
   const person = {
      firstName: 'John',
      lastName: 'Doe',
      fullName: function() {
         return this.firstName + ' ' +
this.lastName;
      }
   };
   console.log(person.fullName()); //
Outputs: John Doe
   ```

These examples cover a wide range of topics in HTML, CSS, and JavaScript, providing practical demonstrations of concepts and techniques. If you have more specific questions or need further explanation on any topic, feel free to ask!
Certainly! Let's continue exploring more topics:

61. **`<section>` vs `<div>`:**
   - `<div>`: Used as a generic container for grouping and styling content.
   - `<section>`: Represents a thematic grouping of content, typically with a heading.

   Example:
   ```html
   <div class="container">
      <div>Generic content goes
here.</div>
   </div>

   <section>
      <h2>Section Title</h2>
      <p>Thematic content goes
here.</p>
   </section>
   ```

62. **Centering Elements Horizontally and Vertically:**
   Centering elements in CSS can be achieved using various techniques such as Flexbox, CSS Grid, absolute positioning with translate, and margin auto.

   Example (Flexbox):
   ```html
   <style>
      .container {
         display: flex;
         justify-content: center;
         align-items: center;
         height: 300px;
      }

      .centered {
         width: 200px;
         height: 100px;
         background-color: lightblue;
      }
   </style>
   <div class="container">
      <div
class="centered">Centered
content</div>
   </div>
   ```

63. **JavaScript Spread and Rest Operators:**
   - Spread Operator (`...`): Allows an iterable (like an array or string) to be expanded into individual elements.
   - Rest Parameter (`...`): Allows a function to accept an indefinite number of arguments as an array.

   Example (Spread Operator):
   ```javascript
   const numbers = [1, 2, 3];
   const newNumbers = [...numbers,
4, 5];
   console.log(newNumbers); //
Outputs: [1, 2, 3, 4, 5]
   ```

   Example (Rest Parameter):
   ```javascript
   function sum(...args) {
      return args.reduce((acc, val) =>
acc + val, 0);
   }
   console.log(sum(1, 2, 3, 4, 5)); //
Outputs: 15
   ```

64. **Responsive Layouts with HTML (meta viewport):**
   The `<meta>` viewport tag in HTML allows web developers to control the viewport's width and scale on different devices, ensuring proper responsiveness.

   Example:
   ```html
   <meta name="viewport"
content="width=device-width,
initial-scale=1.0">
   ```

65. **CSS Preprocessors (SASS, LESS):**
   CSS preprocessors like SASS (Syntactically Awesome Style Sheets) and LESS (Leaner Style Sheets) extend CSS with variables, nesting, mixins, and other advanced features, which are compiled into standard CSS.

   Example (SASS):
   ```scss
   $primary-color: #3498db;
   $secondary-color: #e74c3c;

   .button {
      background-color:
$primary-color;
      color: white;
      &:hover {
         background-color:
$secondary-color;
      }
   }
   ```

66. **JavaScript Destructuring Assignment:**
   Destructuring assignment in JavaScript allows you to extract

values from arrays or objects into distinct variables, making it easier to work with complex data structures.

   Example (Array Destructuring):
```javascript
   const numbers = [1, 2, 3];
   const [a, b, c] = numbers;
   console.log(a, b, c); // Outputs: 1 2
3
```

   Example (Object Destructuring):
```javascript
   const person = { firstName: 'John',
lastName: 'Doe' };
   const { firstName, lastName } =
person;
   console.log(firstName, lastName);
// Outputs: John Doe
```

67. **`<!DOCTYPE html>` Declaration:**
   The `<!DOCTYPE html>` declaration at the beginning of an HTML document specifies the version of HTML used (HTML5 in this case) and triggers standards mode rendering in browsers.

   Example:
```html
   <!DOCTYPE html>
   <html lang="en">
   <head>
      <meta charset="UTF-8">
      <title>Document</title>
   </head>
   <body>
      Content goes here...
   </body>
   </html>
```

68. **Including Custom Fonts (@font-face, Google Fonts):**
   Custom fonts can be included in web projects using `@font-face` for self-hosted fonts or using services like Google Fonts for easy integration of web fonts.

   Example (Google Fonts):
```html
   <link
href="https://fonts.googleapis.com/cs
s2?family=Roboto:wght@400;700&d
isplay=swap" rel="stylesheet">
   <style>
      body {
         font-family: 'Roboto',
sans-serif;
      }
   </style>
```

69. **JavaScript Date and Time:**
   JavaScript provides built-in objects and methods for working with dates and times, allowing manipulation, formatting, and calculation of dates.

   Example:
```javascript
   const now = new Date();

console.log(now.toLocaleDateString(
)); // Outputs: current date in locale
format
```

70. **Handling Broken Images (`onerror` attribute):**
   The `onerror` attribute in HTML allows you to specify JavaScript code to run if an `<img>` element fails to load.

   Example:
```html
   <img src="nonexistent.jpg"
onerror="this.src='fallback.jpg';">
```

These examples cover various aspects of HTML, CSS, and JavaScript, providing practical demonstrations and explanations. If you have more questions or need further clarification on any topic, feel free to ask!
Certainly! Let's continue exploring more topics:

71. **Handling Browser Compatibility (Vendor Prefixes, Feature Queries, Polyfills):**
   Ensuring cross-browser compatibility involves:
   - **Vendor Prefixes:** Using browser-specific prefixes (-webkit-, -moz-, -ms-, -o-) for CSS properties to ensure compatibility with different browsers.
```css
   .box {
      -webkit-border-radius: 5px;
      -moz-border-radius: 5px;
      border-radius: 5px;
   }
```
   - **Feature Queries:** Using `@supports` in CSS to apply styles only if a certain feature is supported by the browser.
```css
   @supports (display: grid) {
      .container {
         display: grid;
         grid-template-columns: 1fr
1fr;
      }
   }
```
   - **Polyfills:** JavaScript code that provides modern functionality on older browsers that do not natively support it.
```html
   <script
src="https://cdn.polyfill.io/v3/polyfill.
min.js"></script>
```

72. **JavaScript Math and Number Methods:**
   JavaScript provides built-in Math methods for mathematical operations and Number methods for numeric operations and conversions.
   - **Math Methods:**
```javascript
   console.log(Math.sqrt(25)); //
Outputs: 5 (square root)
```
   - **Number Methods:**
```javascript
   const num = 10.456;
   console.log(num.toFixed(2)); //
Outputs: "10.46" (toFixed)
```

73. **CSS Frameworks (Bootstrap, Foundation):**
   CSS frameworks like Bootstrap and Foundation provide pre-designed and pre-styled UI components and layouts for easier and faster web development.
   - **Bootstrap Example:**
```html
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.
com/bootstrap/4.5.0/css/bootstrap.mi
n.css">
   <div class="container">
      <button class="btn
btn-primary">Primary
Button</button>
   </div>
```

74. **JavaScript Definition and Purpose:**
   JavaScript is a high-level, interpreted programming language used primarily for creating interactive and dynamic content on web pages.
   - **Example of Inline JavaScript:**
```html
   <script>
      alert('Hello, JavaScript!');
   </script>
```

75. **CSS-in-JS (Styled-components, Emotion):**
   CSS-in-JS libraries like Styled-components and Emotion allow developers to write CSS directly in JavaScript, enabling component-level styling and scoped CSS.
   - **Styled-components Example:**
```javascript
   import styled from
'styled-components';
```

```
   const Button = styled.button`
      background-color: ${props => props.primary ? 'blue' : 'gray'};
      color: white;
      padding: 10px 20px;
   `;
```

76. **Front-End Build Tools (Webpack, Gulp, Grunt):**
   Front-end build tools automate tasks like bundling, minifying, and optimizing code, enhancing development workflow and performance.
   - **Webpack Example:**
```javascript
   module.exports = {
      entry: './src/index.js',
      output: {
         filename: 'bundle.js',
         path: path.resolve(__dirname, 'dist'),
      },
      module: {
         rules: [
            {
               test: /\.css$/,
               use: ['style-loader', 'css-loader'],
            },
         ],
      },
   };
```

77. **Version Control with Git:**
   Git is a distributed version control system used for tracking changes in source code during software development.
   - **Basic Git Commands:**
```bash
   git init
   git add .
   git commit -m "Initial commit"
   git push origin master
```

78. **Package Managers (npm, Yarn):**
   Package managers like npm (Node Package Manager) and Yarn manage dependencies and facilitate package installation for JavaScript projects.
   - **npm Example:**
```bash
   npm install lodash
```

79. **Front-End Testing (Jest, Mocha, Chai):**
   Testing frameworks like Jest, Mocha, and assertion libraries like Chai are used for writing and executing automated tests in front-end development.
   - **Jest Example:**
```javascript
   test('adds 1 + 2 to equal 3', () => {
      expect(sum(1, 2)).toBe(3);
   });
```

80. **Responsive Design Principles:**
   Principles such as fluid grids, flexible images, and media queries are used to create web designs that adapt to different screen sizes and devices.

81. **Progressive Web Apps (PWAs):**
   PWAs use modern web capabilities to provide an app-like experience, including offline functionality, push notifications, and more.
   - **Example:**
```html
   <link rel="manifest" href="/manifest.json">
```

82. **Single Page Applications (SPAs):**
   SPAs load dynamically and update content without refreshing the entire page, typically using frameworks like React or Angular.

83. **RESTful APIs and JSON:**
   RESTful APIs use HTTP requests to perform CRUD (Create, Read, Update, Delete) operations, typically exchanging data in JSON format.

84. **GraphQL Basics:**
   GraphQL is a query language for APIs that enables clients to request exactly the data they need, simplifying data fetching and reducing over-fetching.

85. **Web Performance Optimization:**
   Techniques like minification, lazy loading, caching, and optimizing assets are used to improve website loading speed and user experience.

86. **Cross-Browser Compatibility Issues:**
   Issues arise due to variations in browser rendering engines and support for web standards, requiring testing and sometimes specific workarounds.

87. **Browser DevTools Usage:**
   Browser developer tools provide debugging, profiling, and editing capabilities to inspect and manipulate HTML, CSS, and JavaScript in real-time.

88. **Accessibility Best Practices:**
   Practices like semantic HTML, ARIA roles, keyboard navigation, and color contrast ensure web content is accessible to users with disabilities.

89. **Web Security Basics (CORS, XSS, CSRF):**
   Understanding concepts like Cross-Origin Resource Sharing (CORS), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) is crucial for securing web applications.

90. **SEO Basics for Developers:**
   Techniques like using meaningful HTML tags, meta tags, proper URL structure, and optimizing content for keywords help improve search engine rankings.

91. **Code Versioning Best Practices:**
   Best practices include using descriptive commit messages, branching strategies, and code reviews to maintain code quality and collaboration.

92. **CI/CD Pipelines:**
   Continuous Integration (CI) and Continuous Deployment (CD) pipelines automate the process of testing and deploying code changes.

93. **Front-End Performance Testing:**
   Tools like Lighthouse, WebPageTest, and GTmetrix are used to measure and optimize front-end performance metrics like page load time and responsiveness.

94. **Front-End Component Libraries (React, Angular, Vue):**
   Component-based libraries and frameworks like React, Angular, and Vue enable building scalable and reusable UI components.

95. **State Management in Front-End Frameworks (Redux, Vuex):**
   State management libraries like Redux (for React) and Vuex (for Vue) manage global state in complex applications to maintain data consistency.

96. **Web Sockets and Real-Time Communication:**
   Web Sockets enable bidirectional communication between clients and servers, facilitating real-time updates in web applications.

97. **Service Workers and Web Workers:**

Service Workers manage background tasks like push notifications and offline caching, while Web Workers run scripts in background threads.

98. **Client-Side Routing (React Router, Vue Router):**
Client-side routing libraries like React Router and Vue Router handle navigation and state management within SPAs.

99. **HTML Semantic Elements:**
Semantic HTML elements like `<header>`, `<nav>`, and `<footer>` provide meaning to the structure of a web page for improved accessibility and SEO.

100. **CSS Grid vs Flexbox:**
CSS Grid and Flexbox are layout systems in CSS:
- **Flexbox:** One-dimensional layout for arranging items in a row or column.
- **CSS Grid:** Two-dimensional layout for arranging items in rows and columns.

Congratulations on reaching the end of the list! If you have more questions or need further explanations on any topic, feel free to ask!