HTML

1. HTML Introduction

HTML (HyperText Markup Language) is the standard language used to create and design webpages. It structures the content using elements or tags that define various types of content like headings, paragraphs, images, links, etc. HTML files are saved with the `.html` extension and can be viewed in any web browser. It was developed to facilitate information sharing over the web and supports hyperlinks that allow navigation between pages.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML Introduction</title>
</head>
<body>
  <h1>Welcome to HTML!</h1>
  <p>This is a paragraph in HTML.</p>
</body>
</html>
```

2. HTML Basic Examples

In an HTML document, you always begin with a `<!DOCTYPE>` declaration to specify the document type and version (HTML5 in most modern cases). The document is structured with tags inside a pair of `<html>` tags. The `<head>` tag contains metadata and links to resources (e.g., stylesheets, scripts), while the `<body>` tag contains the visible content of the page (text, images, links, etc.).

3. HTML Elements

HTML elements are the building blocks of an HTML document. An element consists of a start tag (e.g., `<h1>`), content (e.g., "Hello, World!"), and an end tag (e.g., `</h1>`). Some elements are self-closing, like `<img>` and `<br>`. Each element serves a specific purpose, such as displaying a heading, paragraph, or image.

Example:

```
<h1>This is a heading</h1>
<p>This is a paragraph element.</p>
<a href="https://www.example.com">This is a link</a>
```

4. HTML Attributes

Attributes provide additional information about an HTML element. They are always included within the opening tag and usually come in name/value pairs. For instance, in `<a href="https://www.example.com">Example</a>`, the `href` attribute specifies the URL the link points to. Common attributes include `id`, `class`, `src`, `alt`, `style`, etc.

Example:

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

<img src="image.jpg" alt="Description of image" width="300" height="200">
```

5. HTML Headings

HTML headings define titles or subtitles in a document, using tags from `<h1>` to `<h6>`. The `<h1>` tag defines the most important heading, often used for main titles, while `<h6>` is the least important. Headings help organize content and improve SEO (Search Engine Optimization) by defining a clear content hierarchy.

Example:

```
<h1>Heading 1</h1>

<h2>Heading 2</h2>

<h3>Heading 3</h3>
```

6. HTML Paragraphs

Paragraphs are defined using the `<p>` element. Each paragraph is rendered on a new line in the browser. This tag is primarily used to format blocks of text content in HTML documents, making text more readable by separating it into chunks.

Example:

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

<p>This is another paragraph.</p>
```

7. HTML Text Formatting

HTML offers several tags for text formatting, allowing you to apply styling directly to text. Common text formatting tags include:

- `<b>` for bold text,
- `<i>` for italic text,
- `<u>` for underlined text,
- `<sup>` for superscript,
- `<sub>` for subscript.

These tags help enhance the visual presentation of content.

Example:

```
<p>This is <b>bold</b> text and this is <i>italic</i> text.</p>

<p>This is <u>underlined</u> text.</p>
```

8. HTML Links

Hyperlinks are one of the key features of HTML. The `<a>` (anchor) tag is used to create links. The `href` attribute within this tag specifies the destination URL. Links can lead to other pages, files, sections of the same page, or external resources. Additionally, attributes like `target="_blank"` allow links to open in a new tab.

Example:

```
<a href="https://www.example.com">Click here to visit Example</a>
```

9. HTML Images

Images are embedded into an HTML page using the `<img>` tag. This tag requires the `src` (source) attribute to specify the image file location and the `alt` attribute to provide alternative text if the image cannot load. Other attributes like `width` and `height` control the display size of the image. Images make content more engaging and visually appealing.

Example:

```
<img src="image.jpg" alt="An example image" width="300" height="200">
```

10. HTML Favicon

A favicon (short for "favorite icon") is a small image displayed in the browser tab beside the webpage title. It helps in branding and visual identification of the website. The favicon is typically a 16x16 or 32x32 pixel image and is specified within the `<head>` section using a `<link>` tag.

Example:

```
<head>
  <link rel="icon" href="favicon.ico" type="image/x-icon" ></link>
</head>
```

11. HTML Tables

HTML tables allow you to present data in rows and columns. A table is created using the `<table>` tag. Each row is represented by a `<tr>` (table row), and each cell within a row is defined by either `<th>` (table header) or `<td>` (table data). Tables are commonly used for organizing structured data like statistics or schedules, though CSS grids and flexboxes are now preferred for layout purposes.

Example:

```
<table>
  <tr>
    <th>Header 1</th>
    <th>Header 2</th>
  </tr>
  <tr>
    <td>Data 1</td>
    <td>Data 2</td>
  </tr>
</table>
```

12. HTML Lists

HTML supports ordered (`<ol>`) and unordered (`<ul>`) lists for presenting items in a list format.

- Ordered lists are numbered, whereas unordered lists use bullet points.
- Each list item is represented by the `<li>` (list item) tag. Lists are useful for grouping related items or steps in a sequence.

Example:

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

```
</ul>

<ol>
  <li>First Item</li>
  <li>Second Item</li>
</ol>
```

13. HTML Block and Inline Elements

HTML elements are classified into two categories:

- **Block elements**: These elements take up the full width of their container and always start on a new line (e.g., `<div>`, `<h1>`, `<p>`).
- **Inline elements**: These elements take up only as much width as needed and appear alongside other elements (e.g., `<span>`, `<a>`, `<img>`). Understanding the difference is essential for designing page layouts.

Example:

```
<div>This is a block element.</div>

<span>This is an inline element.</span>
```

14. HTML Iframes

An iframe (inline frame) is used to embed another HTML document within the current webpage. The `<iframe>` tag is typically used for embedding external content like videos, maps, or other websites. The `src` attribute defines the source URL of the embedded content.

Example:

```
<iframe src="https://www.example.com" width="300" height="200"></iframe>
```

15. HTML File Paths

File paths tell the browser where to find files like images, videos, or stylesheets.

- **Absolute paths** specify the full URL to the resource (e.g., `https://www.example.com/image.jpg`).
- **Relative paths** define the path in relation to the current HTML file (e.g., `images/photo.jpg`). Correct file paths ensure that resources load properly.

Example:

```
<img src="images/photo.jpg" alt="Photo">
```

16. HTML - The Head Element

The `<head>` element contains metadata about the document that isn't visible on the webpage but is important for search engines, browser behavior, and document identification. Inside the `<head>` tag, you can include elements such as:

- `<title>`: Defines the title of the webpage displayed in the browser tab.
- `<meta>`: Provides metadata like charset, author, or viewport settings for mobile responsiveness.
- `<link>`: Links to external resources like stylesheets or favicons.

Example:

```
<head>
  <title>Page Title</title>
  <meta charset="UTF-8">
</head>
```

17. HTML Layout Elements and Techniques

HTML5 introduced new layout elements to semantically organize content:

- `<header>`: Represents the header of a page or section.
- `<nav>`: Defines a navigation bar for links.
- `<section>`: Represents a standalone section of content.
- `<article>`: Represents a self-contained article or post.
- `<footer>`: Represents the footer of a page. These elements help create a structured and accessible layout for the webpage.

Example:

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

```
</nav>
<section>
  <h2>Section Heading</h2>
  <p>Section content here.</p>
</section>
<footer>
  <p>Footer content.</p>
</footer>
```

18. HTML Computer Code Elements

HTML provides specific tags for displaying programming code in documents:

- `<code>`: Used to display inline code snippets.
- `<pre>`: Preserves the formatting of text and displays it in a fixed-width font (preformatted).
- `<kbd>`: Used to display keyboard input.

Example:

```
<pre>
def hello():
  print("Hello, World!")
</pre>
```

19. HTML Semantic Elements

Semantic HTML elements clearly describe their meaning to both the browser and developers. Examples include:

- `<header>`, `<footer>`, `<nav>`, `<article>`, `<aside>`, and `<section>`. These elements improve the accessibility and readability of web content for both users and search engines.

Example:

```
<article>
  <h2>Article Title</h2>
  <p>Article content goes here.</p>
</article>
```

20. HTML Entities

HTML entities allow you to display reserved characters (like <, >, or &) or special characters (e.g., ©, ™). They are written using a name or a numerical reference.

Example:

```
<p>This is how you display the less-than symbol: &lt;</p>
```

21. HTML Symbols

HTML supports many types of symbols, including mathematical operators and special characters, that can be rendered using their entity codes. For instance, `&copy;` displays the copyright symbol (©), and `&euro;` displays the euro sign (€).

Example:

```
<p>&#169; 2024</p>
```

22. Using Emojis in HTML

Emojis can be inserted into HTML documents either using their Unicode (e.g., `&#128512;` for 😀) or simply by copying and pasting the emoji character. Modern browsers support emoji rendering.

Example:

```
<p>😊 This is a smiley face emoji.</p>
```

23. HTML Encoding (Character Sets)

Character encoding determines how characters are represented in a document. UTF-8 is the most widely used encoding because it supports almost all characters and symbols in the world. It is defined using the `<meta charset="UTF-8">` tag in the document's `<head>` section.

Example:

```
<meta charset="UTF-8">
```

24. HTML Forms

Forms collect user input and send it to a server for processing. The `<form>` tag includes form elements like text fields (`<input type="text">`), radio buttons, checkboxes, submit buttons, and text areas. The form's action attribute specifies the URL to send the data to, and the method attribute defines how the data is sent (usually `POST` or `GET`).

Example:

```
<form action="/submit" method="post">
  <label for="name">Name:</label>
  <input type="text" id="name" name="name">
  <input type="submit" value="Submit">
</form>
```

25. HTML Media

HTML5 supports embedding media elements like audio and video natively without the need for external plugins (e.g., Flash).

- **Video**: Embedded using the `<video>` tag, supporting attributes like `controls`, `autoplay`, and `loop`.
- **Audio**: Embedded using the `<audio>` tag with similar attributes. These elements allow you to embed rich multimedia content.

Example:

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  Your browser does not support the video tag.
</video>
```

CSS

1. CSS Introduction

CSS (Cascading Style Sheets) is a stylesheet language used for describing the presentation of a document written in HTML. CSS controls the layout, formatting, and overall appearance of web pages, allowing you to style text, images, tables, and other content. It provides flexibility and control over various aspects such as colors, fonts, spacing, and positioning. CSS is essential for creating visually appealing websites and ensuring responsive design across devices.

Example:

```
<style>
  h1 {
    color: blue;
    font-size: 36px;
  }
</style>
```

2. CSS Syntax

CSS syntax is a set of rules for defining the styles of HTML elements. It consists of selectors (which target HTML elements) and declarations (which define the styles for the targeted elements). Declarations are enclosed in curly braces and contain property-value pairs, where properties define the aspects to style (like color or font-size), and values define how to style them. For example, `color: blue;` is a declaration.

Syntax:

```
selector {
  property: value;
}
```

Example:

```
p {
  color: red;
  font-size: 16px;
}
```

3. CSS Selectors

Selectors are patterns used to target HTML elements that you want to style. Different types of selectors include:

- **Element selectors**: Target specific HTML elements like `<p>`, `<div>`, or `<h1>`.
- **Class selectors**: Target elements with a specified class using a period (`.`) followed by the class name (e.g., `.class-name`).
- **ID selectors**: Target a single element with a specific ID using a hash (`#`) followed by the ID (e.g., `#id-name`). Selectors are fundamental in applying styles efficiently to various elements in an HTML document.

Example:

```
/* Element selector */
h1 {
  color: green;
}

/* Class selector */
.box {
  border: 1px solid black;
}

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

4. How to Add CSS

CSS can be added to HTML in three main ways:

- **Inline CSS**: Written directly inside the `style` attribute of HTML elements. It is used for quick, single-instance styling.
- **Internal CSS**: Placed within a `<style>` tag inside the `<head>` section of an HTML document. It is used for styling a single document.
- **External CSS**: Stored in a separate `.css` file and linked to HTML documents using the `<link>` tag. This method is best for styling multiple pages as it keeps the style rules centralized and reusable across documents.

Example:

```
<!-- Inline -->
<p style="color: red;">This is a red paragraph.</p>

<!-- Internal -->
<style>
  body {
    background-color: lightgray;
```

```
  }
</style>

<!-- External (in a separate CSS file) -->
<link rel="stylesheet" href="styles.css">
```

5. CSS Colors

CSS allows you to apply colors to elements using different methods such as:

- **Named colors** (e.g., red, blue, green),
- **HEX values** (e.g., `#ff0000`),
- **RGB values** (e.g., `rgb(255, 0, 0)`), and
- **HSL values** (e.g., `hsl(0, 100%, 50%)`). CSS color properties control the text color (`color`), background color (`background-color`), and other element-specific colors (like borders).

Example:

```
h1 {
  color: blue;       /* Named color */
}

p {
  color: #ff0000;    /* HEX color */
}

div {
  color: rgb(0, 255, 0);  /* RGB color */
}

span {
  color: hsl(240, 100%, 50%); /* HSL color */
}
```

6. CSS Backgrounds

CSS background properties control the appearance of an element's background. You can set background colors, images, positions, sizes, and whether an image repeats or not. Common properties include:

- **`background-color`**: Specifies the background color.
- **`background-image`**: Sets an image as the background.
- **`background-position`**: Controls where the background image is positioned.

- **`background-size`**: Resizes the background image.
- **`background-repeat`**: Determines if and how the image repeats.

Example:

```
body {
  background-color: lightblue;
  background-image: url('background.jpg');
  background-repeat: no-repeat;
  background-size: cover;
}
```

7. Box Model: CSS Borders

The border property in CSS defines the boundary around an HTML element. It is part of the box model, which includes content, padding, borders, and margins. Borders can be customized by setting their width, style (solid, dotted, dashed), and color. CSS borders can also be made to have rounded corners using the `border-radius` property.

Example:

```
div {

  border: 2px solid black;

}
```

8. CSS Margins

Margins define the space outside the element's border, separating it from other elements. You can set the margins for all sides or individually (top, right, bottom, left). Margin values can be auto, percentage, or length units like pixels (`px`) or ems. The CSS margin property helps in controlling the layout and spacing of elements in a web page.

Example:

```
p {
  margin: 20px;
}

h1 {
  margin-top: 10px;
  margin-bottom: 15px;
}
```

9. CSS Padding

Padding refers to the space between the content of an element and its border. It helps to create internal space around the content. Like margins, padding can be specified for all sides or each side individually. Padding is part of the box model and affects the overall size of the element.

Example:

```
div {
  padding: 10px;
}
```

10. CSS Height, Width, and Max-width

Height and width properties in CSS control the dimensions of an element. They can be defined using fixed values (like `px`, `%`, or `em`). The `max-width` property ensures that an element does not exceed a specified width, making it useful for responsive designs where the element's size should adapt to different screen sizes.

Example:

```
div {
  width: 300px;
  height: 200px;
  max-width: 100%;
}
```

11. CSS Outline

The outline property in CSS draws a line around an element, similar to a border but outside of the element’s box model. The outline doesn’t affect the element’s size and doesn’t take up space like borders. You can control the outline’s color, style, and width, but it doesn’t support rounded corners like borders.

Example:

```
p {
  outline: 2px dotted red;
}
```

12. CSS Text

CSS provides a variety of text properties that allow you to control the appearance of text within HTML elements. These properties include:

- **`color`**: Changes the text color.
- **`text-align`**: Aligns text horizontally (left, right, center, justify).
- **`text-decoration`**: Adds decoration like underlines, overlines, or strike-through.
- **`text-transform`**: Controls capitalization (uppercase, lowercase).
- **`letter-spacing`** and **`line-height`**: Control the space between letters and the height of lines.

`Example:`

```
h1 {
  color: blue;
  text-align: center;
  text-transform: uppercase;
}
```

13. CSS Fonts

CSS allows customization of fonts using properties like:

- **`font-family`**: Defines the font type (e.g., Arial, Times New Roman).
- **`font-size`**: Specifies the size of the font (e.g., 16px, 1em).
- **`font-weight`**: Defines the boldness of the text (normal, bold, etc.).
- **`font-style`**: Controls italic or oblique text styles. These properties control the typographical appearance of text on web pages, ensuring readability and aesthetic appeal.

Example:

```
p {
  font-family: 'Arial', sans-serif;
  font-size: 18px;
  font-weight: bold;
}
```

14. CSS Icons

CSS icons are graphical representations that can be inserted into a web page. You can use icon libraries like Font Awesome, which provide scalable vector icons that can be styled with CSS just like text. Icons are often used for buttons, navigation bars, and user interfaces, making them functional and visually appealing.

Example:

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css">

<i class="fas fa-home"></i>
```

15. CSS Links

CSS enables you to style hyperlinks using pseudo-classes to define different states:

- **`:link`**: Default state of an unvisited link.
- **`:visited`**: State of a link that has been clicked.
- **`:hover`**: Defines the style when a user hovers over the link.
- **`:active`**: Style applied when the link is being clicked. These pseudo-classes allow for better user experience and aesthetics by changing link colors, underlines, and effects during interaction.

Example:

```
a:link {
  color: blue;
}

a:hover {
  color: red;
}
```

16. CSS Layout - Width and Max-width

`width` sets the horizontal size of an element, while `max-width` restricts the element from growing larger than a specified value. This ensures that the layout remains flexible and responsive, especially on different screen sizes. For example, `max-width` is frequently used to create adaptive web designs that fit various device widths without exceeding the available space.

Example:

```
div {
  width: 400px;
  max-width: 100%;
}
```

17. CSS Layout - The position Property

The `position` property defines how an element is placed in relation to the normal document flow. It has several values:

- **`static`**: Default, follows the normal flow.
- **`relative`**: Positioned relative to its normal position.
- **`absolute`**: Positioned relative to the nearest positioned ancestor.
- **`fixed`**: Positioned relative to the browser window, and remains fixed during scrolling. The `position` property is key in creating complex layouts by controlling the exact placement of elements.

Example:

```
div {
  position: absolute;
  top: 50px;
  left: 100px;
}
```

18. CSS Layout - The z-index Property

The `z-index` property controls the stacking order of overlapping elements. Elements with a higher `z-index` appear in front of those with a lower value. It only works on positioned elements (`position: relative`, `absolute`, or `fixed`). This is useful in designing web pages where layering content is needed, such as tooltips, modals, or drop-down menus.

Example:

```
div {
  position: absolute;
```

```
  z-index: 10;
}
```

19. CSS Layout – Overflow

The `overflow` property controls how content that overflows an element's box is handled. It has values like:

- **`visible`**: Overflow is not clipped, and the content is visible outside the box.
- **`hidden`**: Overflow is clipped, and the rest is invisible.
- **`scroll`**: Adds scrollbars to view the clipped content.
- **`auto`**: Automatically adds scrollbars if needed. This property ensures that overflowing content is properly managed, avoiding messy layouts.

Example:

```
div {
  width: 300px;
  height: 100px;
  overflow: scroll;
}
```

20. CSS Layout - Float and Clear

`float` allows an element to float next to another element, typically used to position images or text within a container. Common values include `left` or `right`. The `clear` property ensures that no elements float on the left or right of a specified element. Float and clear properties are frequently used in older layouts, although flexbox and grid layouts are now preferred.

Example:

```
img {
  float: left;
  margin-right: 10px;
}

p {
  clear: both;
```

```
}
```

21. CSS Layout - display: inline-block

`inline-block` allows an element to behave like an inline element (not forcing line breaks) while also allowing block-level properties such as width and height. This makes it easier to control the layout of elements like buttons or small containers without breaking the flow of the document.

Example:

```
div {
  display: inline-block;
  width: 100px;
  height: 100px;
}
```

22. CSS Layout - Horizontal & Vertical Align

Horizontal and vertical alignment refers to positioning elements within a container or relative to each other. For horizontal alignment, the `text-align` property is commonly used for text, while CSS Flexbox or Grid is used for both horizontal and vertical alignment. Vertical alignment, especially of inline elements, can be achieved with the `vertical-align` property, or through Flexbox's `align-items` and `justify-content`.

Example:

```
/* Horizontal alignment */
p {
  text-align: center;
}

/* Vertical alignment */
div {
  display: flex;
  justify-content: center;
  align-items: center;
```

```
  height: 200px;
}
```

23. CSS Combinators

Combinators define relationships between CSS selectors:

- **Descendant combinator ( )**: Targets elements inside another element.
- **Child combinator (>)**: Targets immediate child elements.
- **Adjacent sibling combinator (+)**: Targets the next sibling element.
- **General sibling combinator (~)**: Targets all sibling elements. Combinators allow for more specific and complex styling rules.

Example:

```
/* Descendant */
div p {
  color: blue;
}

/* Child */
div > p {
  color: red;
}
```

24. CSS Pseudo-classes

Pseudo-classes are used to define the special states of an element, such as when a user hovers over it (`:hover`), focuses on a form field (`:focus`), or clicks on a link (`:active`). Other pseudo-classes like `:nth-child()` target specific children of an element. Pseudo-classes enhance interactivity and responsiveness in web designs.

Example:

```
a:hover {
  color: green;
}

p:nth-child(2) {
  color: blue;
```

```
}
```

25. CSS Pseudo-elements

Pseudo-elements are used to style specific parts of an element, like the first letter, first line, or content before and after the element. Common pseudo-elements include `::before`, `::after`, `::first-letter`, and `::first-line`. They are useful for adding decorative content or additional formatting to specific parts of text.

Example:

```
p::first-letter {
  font-size: 2em;
  color: red;
}
```

26. CSS Opacity / Transparency

The `opacity` property in CSS controls the transparency of an element. A value between 0 (fully transparent) and 1 (fully opaque) is applied to the entire element, including its content and background. This property is useful for creating visual effects like faded backgrounds, translucent images, or overlay effects.

Example:

```
div {
  opacity: 0.5;
}
```

27. CSS Navigation Bar

A navigation bar (navbar) is a horizontal or vertical bar on a webpage that contains links to other sections of the site. It is commonly styled using CSS for layout, positioning, and interactivity. Typically, a list (`<ul>`) of links (`<a>`) is used, with CSS properties like `float`, `display`, `padding`, and `hover` effects to create visually appealing and interactive menus.

Example:

```
nav ul {
```

```
  list-style-type: none;
  padding: 0;
}

nav ul li {
  display: inline;
  margin-right: 10px;
}
```

28. CSS Dropdowns

Dropdowns are menus that show a list of options when clicked or hovered over. They are often created using lists (`<ul>`, `<li>`) and styled with CSS, utilizing properties like `position: absolute`, `display: none`, and `:hover` pseudo-classes to control when and how the dropdown appears.

```
nav ul li:hover ul {
  display: block;
}

nav ul ul {
  display: none;
}
```

29. CSS Image Gallery

An image gallery is a collection of images displayed in an organized layout. CSS is used to control the size, spacing, and positioning of the images. Common techniques include using `float` or `flexbox` for aligning the images in rows and columns and adding hover effects to enhance the visual presentation.

Example:

```
.gallery img {
  float: left;
  width: 100px;
  margin: 10px;
}
```

30. CSS Image Sprites

Image sprites combine multiple images into one large image. Instead of loading multiple images, CSS uses the `background-position` property to display only the portion of the image that corresponds to a specific icon or graphic. This reduces server requests and improves performance.

Example:

```
.icon {
  background: url('sprite.png') no-repeat;
}

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

31. CSS Attribute Selectors

Attribute selectors allow you to target HTML elements based on their attributes. They are useful for styling elements with specific attributes (e.g., links with a particular `href`, inputs with a specific `type`, etc.). They can match exact values, partial matches, or even begin or end with a particular substring.

Example:

```
a[href*="example"] {
  color: red;
}
```

32. CSS Forms

CSS provides several properties to style forms and form elements (e.g., inputs, text areas, buttons). You can control their appearance using properties like `padding`, `border`, `background-color`, and `font-size`. Additionally, you can enhance form elements with focus and hover effects to improve user experience.

Example:

```
input[type="text"] {
```

```
  width: 200px;
  padding: 5px;
}
```

33. CSS Units

CSS units define the size of elements, text, margins, and padding. There are two types of units:

- **Absolute units**: Fixed sizes like `px` (pixels), `cm` (centimeters), and `pt` (points).
- **Relative units**: Dynamic sizes based on other elements, such as `%` (percentage), `em` (relative to the font-size of the element), and `rem` (relative to the root font size).

Example:

```
div {
  width: 50%;
  padding: 1em;
}
```

34. CSS Specificity

Specificity determines which CSS rule is applied when multiple rules match the same element. It is calculated based on the types of selectors used (ID selectors have higher specificity than class selectors, and class selectors have higher specificity than element selectors). Inline styles have the highest specificity. Understanding specificity is crucial for resolving conflicts in CSS rules.

Example:

```
/* ID has more specificity */
#header {
  color: blue;
}

/* Class is less specific */
.header {
  color: red;
}
```

35. CSS The !important Rule

The `!important` rule is used to override other styles, regardless of specificity. It forces a particular style to be applied, even if a more specific rule would normally take precedence. While useful, overuse of `!important` can make debugging and maintaining CSS more challenging.

Example:

```
p {
  color: red !important;
}
```

36. CSS Math Functions

CSS includes math functions like `calc()`, `min()`, `max()`, and `clamp()`, which allow developers to perform calculations directly within CSS. These functions provide flexibility by enabling dynamic sizing and responsive layouts without the need for complex CSS code.

Example:

```
div {
  width: calc(100% - 50px);
}
```

Advance CSS

1. CSS Rounded Corners

The border-radius property in CSS is used to create rounded corners for elements, such as boxes, images, and buttons.

Example:

```
.rounded-box {
  width: 200px;
  height: 100px;
  background-color: lightblue;
  border-radius: 15px;
}
```

- **width: 200px; height: 100px;**: Sets the dimensions of the box to 200 pixels wide and 100 pixels tall.
- **background-color: lightblue;**: Sets the background color of the box to light blue.
- **border-radius: 15px;**: Rounds all corners of the box with a radius of 15 pixels. This creates smooth, rounded corners rather than sharp edges.

Individual Corner Control:

```
.rounded-box {
  border-top-left-radius: 10px;
  border-top-right-radius: 20px;
  border-bottom-right-radius: 30px;
  border-bottom-left-radius: 40px;
}
```

Each corner can be adjusted independently:

- **border-top-left-radius: 10px;**: Rounds the top-left corner by 10 pixels.
- **border-top-right-radius: 20px;**: Rounds the top-right corner by 20 pixels.
- **border-bottom-right-radius: 30px;**: Rounds the bottom-right corner by 30 pixels.
- **border-bottom-left-radius: 40px;**: Rounds the bottom-left corner by 40 pixels.

2. CSS Border Images

border-image lets you use an image as the border of an element, rather than a solid color or style like solid or dashed.

Example:

```
.border-image-box {
  border: 10px solid;
  border-image: url('border.png') 30 round;
}
```

- **border: 10px solid;**: Creates a solid border with a thickness of 10 pixels.
- **border-image: url('border.png') 30 round;**: Uses the image located at border.png for the border. The 30 specifies the slice (the distance from each side to slice the image), and round instructs the image to be repeated (tiled) to fill the border.

3. CSS Multiple Backgrounds

CSS allows you to apply multiple background images to a single element by separating each image with a comma.

Example:

```
.multiple-backgrounds {
  background-image: url('image1.png'), url('image2.png');
  background-position: left top, right bottom;
  background-repeat: no-repeat, no-repeat;
}
```

- **background-image: url('image1.png'), url('image2.png');**: Sets two background images. The first background (image1.png) appears below the second (image2.png).
- **background-position: left top, right bottom;**: Positions the first background at the top-left corner and the second at the bottom-right corner.
- **background-repeat: no-repeat, no-repeat;**: Prevents both backgrounds from repeating.

4. CSS Gradients

Gradients allow for smooth color transitions without using an image. There are two main types: linear and radial gradients.

Linear Gradient:

```
.linear-gradient-box {
  background: linear-gradient(to right, red, yellow);
```

```
}
```

- **background: linear-gradient(to right, red, yellow);**: Creates a linear gradient that transitions from red on the left to yellow on the right. The to right parameter indicates the direction of the gradient.

Radial Gradient:

```
.radial-gradient-box {
  background: radial-gradient(circle, red, yellow, green);
}
```

- **background: radial-gradient(circle, red, yellow, green);**: Creates a radial gradient that starts with red in the center, blending outwards to yellow and then to green in a circular pattern.

5. CSS Shadow Effects

CSS provides box-shadow and text-shadow for adding shadow effects to elements and text.

Box Shadow:

```
.shadow-box {
  width: 200px;
  height: 100px;
  background-color: lightblue;
  box-shadow: 5px 5px 15px rgba(0, 0, 0, 0.3);
}
```

- **box-shadow: 5px 5px 15px rgba(0, 0, 0, 0.3);**: Creates a shadow with a 5-pixel horizontal and vertical offset, a 15-pixel blur, and a color of rgba(0, 0, 0, 0.3) (a semi-transparent black).

Text Shadow:

```
.shadow-text {
  font-size: 24px;
  color: darkblue;
  text-shadow: 2px 2px 4px gray;
}
```

- **text-shadow: 2px 2px 4px gray;**: Applies a shadow to the text with a 2-pixel horizontal and vertical offset, a 4-pixel blur, and a gray color.

6. CSS Text Effects

CSS provides properties for text styling, such as text-transform, letter-spacing, and text-decoration.

Example:

```
.text-effect {
  color: #333;
  font-size: 24px;
  text-transform: uppercase;
  letter-spacing: 2px;
  text-decoration: underline;
}
```

- **text-transform: uppercase;**: Converts all text to uppercase.
- **letter-spacing: 2px;**: Adds 2 pixels of space between each letter.
- **text-decoration: underline;**: Underlines the text.

7. CSS Web Fonts

The @font-face rule allows you to use custom fonts in CSS by specifying a font file to be loaded.

Example:

```
@font-face {
  font-family: 'CustomFont';
  src: url('CustomFont.woff2') format('woff2');
}

.custom-font-text {
  font-family: 'CustomFont', sans-serif;
}
```

- **@font-face**: Declares a custom font, specifying the font's name and location.
- **src: url('CustomFont.woff2') format('woff2');**: Loads the font file from a specified URL in WOFF2 format.
- **font-family: 'CustomFont', sans-serif;**: Applies the custom font to an element, with sans-serif as a fallback.

8. CSS 2D Transforms

CSS 2D transforms change the size, position, and rotation of elements.

Example:

```
.transform-box {
  width: 100px;
  height: 100px;
  background-color: coral;
  transform: rotate(45deg) scale(1.2);
}
```

- **transform: rotate(45deg) scale(1.2);**: Rotates the element 45 degrees clockwise and scales it by 1.2 times its original size.

9. CSS Transitions

Transitions animate changes between CSS states, allowing smooth transformations.

Example:

```
.transition-box {
  width: 100px;
  height: 100px;
  background-color: lightgreen;
  transition: background-color 0.5s ease, transform 0.5s ease;
}

.transition-box:hover {
  background-color: darkgreen;
  transform: scale(1.5);
}
```

- **transition: background-color 0.5s ease, transform 0.5s ease;**: Specifies that changes to background-color and transform will animate over 0.5 seconds with a smooth easing function.
- **:hover state**: Changes the background color and scales the box when hovered.

10. CSS Animations

CSS animations enable complex sequences of animations using @keyframes.

Example:

```
@keyframes slideIn {
  0% { transform: translateX(-100%); }
```

```
  100% { transform: translateX(0); }
}

.animation-box {
  width: 100px;
  height: 100px;
  background-color: lightcoral;
  animation: slideIn 2s ease-in-out;
}
```

- **@keyframes slideIn**: Defines the stages of the animation, moving the element from off-screen (-100%) to its normal position (0).
- **animation: slideIn 2s ease-in-out;**: Applies the animation, setting it to last 2 seconds.

11. CSS Styling Images

CSS properties like border-radius, box-shadow, and opacity can style images.

Example:

```
.styled-image {
  width: 300px;
  border-radius: 15px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.5);
  opacity: 0.8;
}
```

- **border-radius: 15px;**: Rounds the corners of the image.
- **box-shadow: 0 4px 8px rgba(0, 0, 0, 0.5);**: Adds a shadow beneath the image.
- **opacity: 0.8;**: Reduces opacity, making the image slightly transparent.

12. CSS object-fit Property

object-fit controls how images or videos fit their container.

Example:

```
.image-fit {
  width: 300px;
  height: 200px;
  object-fit: cover;
}
```

- **object-fit: cover;**: Ensures the image covers the container without distorting its aspect ratio.

13. CSS object-position Property

object-position specifies the alignment of content within the container when object-fit is used.

Example:

```
.image-position {
  width: 300px;
  height: 200px;
  object-fit: cover;
  object-position: top right;
}
```

- **object-position: top right;**: Aligns the image content to the top-right corner of the container.

14. CSS Buttons

CSS allows you to style buttons, adding hover effects and animations.

Example:

```
.button {
  padding: 10px 20px;
  background-color: dodgerblue;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s;
}

.button:hover {
  background-color: deepskyblue;
}
```

- **padding: 10px 20px;**: Adds space inside the button.
- **border-radius: 5px;**: Rounds button corners.
- **transition: background-color 0.3s;**: Smooth color change on hover.

15. CSS Multiple Columns

CSS column-count and column-gap properties divide text into multiple columns.

Example:

```
.multi-column {
  column-count: 3;
  column-gap: 20px;
}
```

- **column-count: 3;**: Splits the content into 3 columns.
- **column-gap: 20px;**: Adds a 20-pixel gap between columns.

16. CSS User Interface

CSS includes cursor, resize, and outline to enhance interactivity.

Example:

```
.ui-element {
  cursor: pointer;
  resize: both;
  outline: 2px solid blue;
}
```

- **cursor: pointer;**: Changes the cursor to indicate a clickable element.
- **resize: both;**: Allows resizing horizontally and vertically.

17. CSS Variables - The var() Function

CSS variables store values that can be reused throughout the stylesheet.

Example:

```
:root {
  --primary-color: teal;
  --spacing: 10px;
}

.variable-box {
  background-color: var(--primary-color);
  margin: var(--spacing);
  padding: var(--spacing);
```

```
}
```

- **:root**: Defines variables at the root level, making them available globally.
- **var(--primary-color);**: Accesses and applies the variable value.

18. CSS Box Sizing

The box-sizing property includes padding and border within an element's total width and height.

Example:

```
.box-sizing-box {
  width: 200px;
  padding: 10px;
  border: 5px solid black;
  box-sizing: border-box;
}
```

- **box-sizing: border-box;**: Keeps the total element width at 200 pixels by including padding and border, simplifying layout management.

CSS Flexbox and Grid

CSS Flexbox

Flexbox is a one-dimensional layout model, meaning it arranges items in either a row (horizontal) or a column (vertical) layout. Flexbox makes it easy to distribute space between items and align them in a flexible way, especially when the layout needs to adapt to different screen sizes.

1. Flex Container

To activate Flexbox, the parent element is designated as a flex container by setting its display property to flex. This allows the child elements (flex items) to be controlled with Flexbox properties.

```
.container {
  display: flex;
}
```

- **Explanation**: display: flex; tells the browser that the .container element will use Flexbox to organize its children. The children of this container (the flex items) will align themselves according to other Flexbox properties that we specify.

2. Flex Items

Flex items are the child elements of the flex container. Each flex item inherits the flex container's properties and can be individually controlled using various Flexbox properties.

```
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

- **Explanation**: The div elements with the class item are children of the .container flex container, making them flex items. Flex items automatically align in a row by default, based on Flexbox's row layout.

3. Flex Direction

The flex-direction property defines the direction the flex items are laid out within the flex container. Flex direction can be:

- row: Items are placed from left to right.
- row-reverse: Items are placed from right to left.
- column: Items are placed from top to bottom.
- column-reverse: Items are placed from bottom to top.

```
.container {
  display: flex;
  flex-direction: row;
}
```

- **Explanation**: Here, flex-direction: row; arranges the items in a row, from left to right. If we change this to column, the items would stack vertically.

4. Flex Wrap

The flex-wrap property controls whether the flex items should wrap onto multiple lines if there isn't enough space in one line. Options include:

- nowrap (default): Items remain on a single line.
- wrap: Items will wrap to a new line if they exceed the container width.
- wrap-reverse: Items will wrap onto multiple lines in reverse order.

```
.container {
  display: flex;
  flex-wrap: wrap;
}
```

- **Explanation**: flex-wrap: wrap; allows items to move onto the next line if they exceed the width of the container. This is helpful for responsive designs where items need to adapt to varying screen sizes.

5. Flex Flow

The flex-flow property is a shorthand for setting flex-direction and flex-wrap in one line.

```
.container {
  display: flex;
  flex-flow: row wrap;
}
```

- **Explanation**: flex-flow: row wrap; specifies that the items should be arranged in a row, and they should wrap onto a new line if they run out of space.

6. Justify Content

The justify-content property aligns the flex items along the main axis (horizontal by default) in the container. Common values are:

- flex-start (default): Items are aligned to the start.
- flex-end: Items are aligned to the end.
- center: Items are centered.
- space-between: Items are evenly spaced, with the first and last items at the edges.
- space-around: Items have equal space around them.

```
.container {
  display: flex;
  justify-content: space-between;
}
```

- **Explanation**: justify-content: space-between; distributes the items with equal space between them, while the first item aligns with the start of the container, and the last item aligns with the end.

7. Align Items

The align-items property aligns flex items along the cross axis (perpendicular to the main axis). Possible values are:

- flex-start: Aligns items to the start of the cross-axis.
- flex-end: Aligns items to the end.
- center: Centers items along the cross-axis.
- baseline: Aligns items based on their text baseline.
- stretch (default): Stretches items to fill the container.

```
.container {
  display: flex;
  align-items: center;
}
```

- **Explanation**: align-items: center; vertically centers the items if the flex direction is row (horizontal layout).

8. Align Content

The align-content property aligns multiple rows of items along the cross-axis when there is extra space (used only with flex-wrap).

```
.container {
  display: flex;
  flex-wrap: wrap;
  align-content: space-around;
}
```

- **Explanation**: align-content: space-around; distributes multiple rows with equal spacing around each row.

9. Order

The order property defines the order in which flex items appear in the container. The default value is 0, and items with lower order values appear first.

```
.item {
  order: 2;
}
```

- **Explanation**: order: 2; moves this item after any items with a lower order. Items are displayed in ascending order based on their order values.

10. Flex Grow

The flex-grow property defines how much a flex item should grow relative to other items. The default value is 0, meaning the item won't grow beyond its default size.

```
.item {
  flex-grow: 1;
}
```

- **Explanation**: flex-grow: 1; allows the item to grow and take up any available space. If multiple items have flex-grow: 1, they share the available space equally.

11. Flex Shrink

The flex-shrink property specifies how much a flex item should shrink if the container's space is limited. A value of 1 (default) means it can shrink; 0 means it won't shrink.

```
.item {
  flex-shrink: 1;
}
```

- **Explanation**: flex-shrink: 1; allows the item to shrink if necessary. Items with higher shrink values will shrink more than items with lower values.

12. Flex Basis

The flex-basis property sets the initial size of the flex item before space distribution. It can be any unit of length (px, %, etc.) or auto.

```
.item {
  flex-basis: 100px;
}
```

- **Explanation**: flex-basis: 100px; sets the item's width (or height in a column layout) to 100 pixels initially. It will grow or shrink based on the available space and the values of flex-grow and flex-shrink.

CSS Grid

CSS Grid is a two-dimensional layout system, meaning it can handle both rows and columns simultaneously, making it ideal for complex layouts like galleries or dashboards.

1. Grid Container

To start using CSS Grid, the container needs display: grid;, which activates Grid properties.

```
.container {
  display: grid;
}
```

- **Explanation**: display: grid; tells the browser to treat .container as a grid, allowing its children (grid items) to follow the grid structure.

2. Grid Items

Grid items are the children of a grid container. Each grid item will occupy a cell in the grid, based on the defined columns and rows.

```
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

3. Grid Columns

The grid-template-columns property defines the number and width of columns in the grid. Each column can have different widths or the same width.

```
.container {
  display: grid;
  grid-template-columns: 100px 200px 100px;
}
```

- **Explanation**: grid-template-columns: 100px 200px 100px; creates three columns: the first is 100 pixels, the second is 200 pixels, and the third is 100 pixels wide.

4. Grid Rows

The grid-template-rows property specifies the number and height of rows in the grid.

```
.container {
  display: grid;
  grid-template-rows: 100px 200px;
}
```

- **Explanation**: grid-template-rows: 100px 200px; creates two rows, the first being 100 pixels tall and the second being 200 pixels tall.

5. Grid Template Areas

The grid-template-areas property allows you to name different grid sections, making placement easier by using the named areas.

```
.container {
  display: grid;
  grid-template-areas:
    "header header"
    "sidebar content"
    "footer footer";
}
.header {
  grid-area: header;
}
```

- **Explanation**: grid-template-areas defines named areas like header, sidebar, and content. The elements assigned to these areas will occupy the defined cells.

6. Grid Template Rows and Columns

grid-template-rows and grid-template-columns let you define the layout structure explicitly with rows and columns.

```
.container {
  display: grid;
  grid-template-rows: 100px 1fr 100px;
  grid-template-columns: 200px 1fr;
}
```

- **Explanation**: Here, three rows are defined: a 100-pixel header, a flexible row, and a 100-pixel footer. Two columns are defined: a 200-pixel sidebar and a flexible main area.

7. Grid Gap

The grid-gap property (or gap) defines the space between grid items, for both rows and columns.

```
.container {
  display: grid;
  grid-gap: 10px;
}
```

- **Explanation**: grid-gap: 10px; creates a 10-pixel gap between all items in the grid.

8. Justify Items

The justify-items property aligns grid items along the row axis (horizontal alignment).

```
.container {
  display: grid;
  justify-items: center;
}
```

- **Explanation**: justify-items: center; centers each grid item horizontally within its cell.

9. Align Items

The align-items property aligns grid items along the column axis (vertical alignment).

```
.container {
  display: grid;
  align-items: center;
```

```
}
```

- **Explanation**: align-items: center; centers each item vertically within its cell.

10. Grid Lines

Grid lines are the lines that divide rows and columns, numbered for referencing positions within the grid.

```
.item {
  grid-column: 1 / 3;
  grid-row: 2 / 4;
}
```

- **Explanation**: grid-column: 1 / 3; spans this item across columns from line 1 to line 3, and grid-row: 2 / 4; spans it across rows from line 2 to line 4.

11. Grid Areas

Grid areas are named sections within the grid, defined with grid-template-areas, allowing easy placement of items.

```
.container {
  display: grid;
  grid-template-areas: 1 / 3 / 2 / 1
    "header header"
    "sidebar content"
    "footer footer";
}
```

By using Flexbox and CSS Grid together, you can create complex and responsive layouts that adapt seamlessly to any screen size, ensuring optimal user experience on any device.

1. Introduction to JavaScript (JS)

JavaScript (JS) is a high-level, interpreted programming language that is primarily used to make web pages interactive. It was created by Netscape and initially named **LiveScript**. JS is often used for both client-side and server-side development.

Advantages:

- **Interactivity**: JavaScript enables interactive web pages, such as forms, pop-ups, animations, etc.
- **Versatility**: It can be used for both front-end (in-browser) and back-end (Node.js) development.
- **Event Handling**: Handles user interactions like clicks, mouse movements, and keyboard inputs.
- **Wide Support**: Supported by all modern browsers.

Disadvantages:

- **Security**: Since it runs in the browser, malicious scripts can exploit security vulnerabilities.
- **Browser Compatibility**: Different browsers may have varying levels of support for JS features.
- **Single-threaded**: JavaScript is single-threaded by default, which can limit performance in certain cases.

2. Internal and External JS

JavaScript can be added to HTML in two ways:

Internal JS:

This is JavaScript code written directly inside an HTML file using <script> tags.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Internal JS Example</title>
</head>
<body>
    <h1>Hello World</h1>
    <script>
        console.log("This is internal JavaScript");
```

```
    </script>
</body>
</html>
```

External JS:

This involves writing the JavaScript in a separate .js file and linking it to the HTML file.

1. Create script.js:

```
console.log("This is external JavaScript");
```

2. Link it to an HTML file:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>External JS Example</title>
</head>
<body>
    <h1>Hello World</h1>
    <script src="script.js"></script>
</body>
</html>
```

3. Introduction to Variables and Basic Programming Concepts in JavaScript

Variables in JavaScript are used to store data values. They can be declared using var, let, or const.

var:

- **Scope**: Function or global scope.
- **Hoisting**: Variables declared using var are hoisted to the top of their scope.

```
var x = 5;
console.log(x);  // Output: 5
```

let:

- **Scope**: Block-level scope.
- **No Hoisting**: Variables declared with let are not hoisted.

```
let y = 10;
console.log(y);  // Output: 10
```

const:

- **Scope**: Block-level scope.
- **Cannot be reassigned**: A const variable cannot be re-assigned after declaration.

```
const z = 15;
console.log(z);  // Output: 15
```

4. Variables and Data Types

In JavaScript, variables can hold different types of data:

- **Primitive Types**: Number, String, Boolean, undefined, null, Symbol, BigInt
- **Reference Types**: Object, Array, Function

```
let name = "Alice";     // String
let age = 25;           // Number
let isStudent = true;   // Boolean
let person = null;      // Null
let data = undefined;   // Undefined
```

5. Conditional Statements

Conditional statements control the flow of execution depending on conditions.

if-else:

```
let age = 20;
if (age >= 18) {
   console.log("You are an adult.");
} else {
   console.log("You are a minor.");
}
```

switch:

```
let fruit = "Apple";
switch (fruit) {
   case "Apple":
      console.log("It's an apple.");
      break;
   case "Banana":
      console.log("It's a banana.");
      break;
   default:
```

```
        console.log("Unknown fruit.");
    }
```

6. Control Flow

Control flow structures like loops and conditional statements allow you to control the execution of code.

7. Functions

Functions are reusable blocks of code that perform a specific task.

```
function greet(name) {
    return "Hello, " + name + "!";
}

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

Arrow Functions:

Arrow functions are a concise way to write functions in JavaScript.

```
const greet = (name) => "Hello, " + name + "!";
console.log(greet("Bob"));  // Output: Hello, Bob!
```

8. Loops

Loops allow repetitive execution of code.

for loop:

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

while loop:

```
let i = 0;
while (i < 5) {
    console.log(i);
    i++;
}
```

forEach loop (for Arrays):

```
const arr = [1, 2, 3];
arr.forEach((item) => {
    console.log(item);  // Output: 1 2 3
});
```

9. Arrays

Arrays are used to store multiple values in a single variable.

```
let colors = ["red", "green", "blue"];
console.log(colors[0]);  // Output: red
```

Accessing Array Elements Using Different Methods:

```
let arr = [10, 20, 30];
console.log(arr[1]);   // Access by index: Output: 20

// Using `forEach`:
arr.forEach((element) => console.log(element));

// Using `map` to transform:
let newArr = arr.map((num) => num * 2);
console.log(newArr);   // Output: [20, 40, 60]
```

10. Objects: Creation and Access

Objects store collections of data in key-value pairs.

Creation:

```
let person = {
    name: "Alice",
    age: 25,
    greet: function() {
        console.log("Hello, " + this.name);
    }
};
console.log(person.name);  // Output: Alice
person.greet();            // Output: Hello, Alice
```

The this Keyword:

The this keyword refers to the current object.

```
let car = {
    brand: "Toyota",
    model: "Corolla",
    getDetails: function() {
        console.log(this.brand + " " + this.model);
    }
};
car.getDetails();  // Output: Toyota Corolla
```

11. The Document Object Model (DOM)

The DOM represents the HTML structure of the web page and provides methods to interact with the page.

Selecting DOM Elements:

```
// Select element by ID
let header = document.getElementById("header");

// Select elements by class name
let items = document.getElementsByClassName("item");

// Select element by querySelector
let button = document.querySelector(".btn");
```

Manipulating DOM Elements:

```
// Change text content
header.textContent = "New Header Text";

// Change style
header.style.color = "red";

// Add event listener
button.addEventListener("click", function() {
    alert("Button clicked!");
});
```

12. Introduction to ES6+ Features

ES6 (ECMAScript 2015) introduced many modern features to JavaScript:

let and const:

- let and const allow block-level scoping, as opposed to var.

Template Literals:

```
let name = "Alice";
let greeting = `Hello, ${name}!`;
console.log(greeting);  // Output: Hello, Alice!
```

Default Parameters:

```
function greet(name = "Guest") {
   console.log("Hello, " + name);
}
greet();  // Output: Hello, Guest
greet("Alice");  // Output: Hello, Alice
```

Destructuring:

Extract values from arrays or objects into variables.

```
let person = { name: "Alice", age: 25 };
let { name, age } = person;
console.log(name);  // Output: Alice
console.log(age);   // Output: 25
```

Modules:

JavaScript modules allow you to break code into smaller files.

```
// In file `math.js`:
export function add(a, b) {
   return a + b;
}

// In file `app.js`:
import { add } from './math.js';
console.log(add(2, 3));  // Output: 5
```