

WEB PROGRAMMING

HYPER TEXT MARK UP LANGUAGE (HTML)

HTML stands for Hypertext Markup Language. It is a programming language used to create and structure web pages. When a web page is created using HTML, it consists of a set of elements and tags that define the structure and content of the page.

The term "Hypertext" in HTML refers to the way web pages are interconnected through links. These links allow users to navigate between different web pages by simply clicking on them. In HTML, links are represented by the `<a>` tag, which stands for anchor.

HTML is considered a markup language because it uses tags to mark up or define different elements within a document. These tags are enclosed in angle brackets (`<` `>`) and provide instructions to web browsers on how to interpret and display the content. For example, the `<h1>` tag is used to mark a heading, the `<p>` tag represents a paragraph, and the `` tag defines an unordered list.

Originally, HTML was developed with the purpose of structuring scientific documents to facilitate the sharing of information among researchers. It provided a standardized way to represent headings, paragraphs, lists, and other elements commonly found in scientific papers.

However, over time, HTML has evolved and become the foundation for formatting and presenting content on the World Wide Web. It is now widely used by web developers to create and design websites. HTML offers a variety of tags and attributes that enable the creation of complex layouts, multimedia integration, forms, and interactive elements.

Web browsers, such as Chrome, Firefox, and Safari, interpret the HTML code and render it as a visual representation of a web page. This allows users to view and interact with the content presented on the web.

In summary, HTML is a markup language used to structure and format web pages. It utilizes tags to define the elements and content within a document, and hypertext allows for the linking of web pages together. HTML has evolved from its scientific origins to become the primary language for creating and formatting websites.

BASIC HTML DOCUMENT

Doctype: The `<!DOCTYPE>` declaration is the very first line of an HTML document. It informs the web browser about the version of HTML being used in the document. The doctype declaration is important because it helps the browser interpret the HTML correctly. The most common doctype declaration for modern HTML documents is `<!DOCTYPE html>`.

HTML: The `<html>` element is the root element of an HTML document. It serves as a container for all the other elements of the page. The opening tag `<html>` is placed at the beginning of the document, and the closing tag `</html>` is placed at the end. All other elements are nested inside the `<html>` element.

Head: The `<head>` element is a container for metadata and other non-visible elements in an HTML document. It is placed immediately after the opening `<html>` tag. The content inside the `<head>` element is not displayed on the web page itself, but it provides important information to the browser and search engines. This is where you would typically include the document's title, links to CSS stylesheets, scripts, and other meta information.

Title: The `<title>` element is used to define the title of an HTML document. It is placed within the `<head>` element. The text placed between the opening `<title>` and closing `</title>` tags appears as the title of the web page in the browser's title bar or tab. Additionally, it is also used by search engines when displaying search results.

Body: The `<body>` element represents the main content area of an HTML document. It is placed immediately after the closing `</head>` tag. All the visible content of the web page, such as text, images, links, and other elements, is placed within the `<body>` element. The opening `<body>` tag is followed by the content, and the closing `</body>` tag is placed at the end of the content.

h1: The `<h1>` element represents the highest level of heading in HTML. It is used to define the main heading or title of a section or webpage. The `<h1>` tag is typically used once per page or section and is often followed by subheadings of lower hierarchy such as `<h2>`, `<h3>`, and so on. The actual text of the heading is placed between the opening `<h1>` and closing `</h1>` tags.

p: The `<p>` element is used to define a paragraph in HTML. It represents a block of text that is typically separated from other paragraphs. The `<p>` tag is used to structure and format text content, and it automatically adds vertical spacing before and after the paragraph. Text content placed between the opening `<p>` and closing `</p>` tags is treated as a paragraph.

In summary, these terms are fundamental elements in HTML:

- `<!DOCTYPE>` declares the HTML version.
- `<html>` is the root element of the HTML document.
- `<head>` contains metadata and non-visible elements.
- `<title>` specifies the title of the document.
- `<body>` represents the visible content area.
- `<h1>` defines the main heading.
- `<p>` represents a paragraph of text.

HTML HEADING TAG

In HTML, heading tags are used to define different levels of headings or titles within a document. These tags range from `<h1>` to `<h6>`, with `<h1>` being the highest level and

`<h6>` the lowest level. The purpose of heading tags is to structure the content and provide visual hierarchy to the headings on a webpage.

Here are the heading tags along with their usage:

1. **`<h1>`:** Represents the highest level of heading. It is typically used for the main title of a page or section. Example: `<h1>Welcome to My Website</h1>`
2. **`<h2>`:** Represents a level below `<h1>` and is used for subheadings. Example: `<h2>About Me</h2>`
3. **`<h3>`:** Represents a lower level heading, often used for subheadings under `<h2>`. Example: `<h3>Education</h3>`
4. **`<h4>`:** Represents a lower level heading, typically used for subheadings under `<h3>`. Example: `<h4>Work Experience</h4>`
5. **`<h5>`:** Represents a lower level heading, used for subheadings under `<h4>`. Example: `<h5>Skills</h5>`
6. **`<h6>`:** Represents the lowest level of heading, used for subheadings under `<h5>`. Example: `<h6>Contact Information</h6>`

When using these heading tags, it is good practice to follow a hierarchical structure, starting with `<h1>` as the main heading and descending down to lower-level headings as needed. This helps organize the content and makes it easier for both humans and search engines to understand the structure and importance of different sections on the page.

THE PARAGRAPH TAG

The `<p>` tag in HTML is used to define a paragraph of text. It represents a block of text that is typically separated from other paragraphs. The `<p>` tag helps structure and format the textual content on a webpage, allowing for better readability and visual organization.

Here's an example of using the `<p>` tag in HTML:

```
html
Copy code

<!DOCTYPE html>
<html>
<head>
  <title>Example Paragraph Tag</title>
</head>
<body>
  <h1>Welcome to My Website</h1>

  <h2>About Me</h2>
  <p>I am a web developer with a passion for coding. I enjoy creating responsive websites and user-friendly interfaces.</p>

  <h3>Education</h3>
  <p>I graduated with a degree in Computer Science. During my studies, I learned the fundamentals of web development and gained hands-on experience through internships and personal projects.</p>

  <h4>Work Experience</h4>
  <p>I have worked for several tech companies, where I developed websites and applications, collaborated with cross-functional teams, and managed client relationships.</p>

  <h5>Skills</h5>
  <p>My skills include HTML, CSS, JavaScript, and other web development technologies. I am also proficient in problem-solving, communication, and teamwork.</p>

  <h6>Contact Information</h6>
  <p>You can reach me at example@example.com. Feel free to contact me for any inquiries or collaborations.</p>
</body>
</html>
```

In this example, we have various sections on the webpage, and each section is introduced by a heading tag. Following each heading, we use the `<p>` tag to define paragraphs that provide more details or information about the respective section.

For instance, under the "About Me" section, we have a paragraph that describes the web developer's passion and expertise. Similarly, the other sections like "Education," "Work Experience," "Skills," and "Contact Information" have corresponding paragraphs that provide relevant information.

The `<p>` tag helps separate and structure the text content, making it easier to read and comprehend. It automatically adds vertical spacing before and after the paragraph, visually distinguishing it from other elements on the page.

By using the `<p>` tag, you can organize and present textual content in a clear and structured manner, enhancing the overall readability and user experience of your webpage.

THE LINE BREAKING TAG

The line breaking tag in HTML is used to create a line break, forcing text or elements to start on a new line. The line breaking tag is represented by the `
` tag. It is a self-closing tag, meaning it doesn't require a closing tag.

Here's an example of using the line breaking tag in HTML:

```
html
Copy code

<!DOCTYPE html>
<html>
<head>
  <title>Example Line Breaking Tag</title>
</head>
<body>
  <h1>Welcome to My Website</h1>

  <p>This is the first line of text.<br>
  This is the second line of text.</p>

  <h2>About Me</h2>
  <p>I am a web developer with a passion for coding.<br>
  I enjoy creating responsive and user-friendly websites.</p>

  <h3>Contact Information</h3>
  <p>You can reach me at example@example.com.<br>
  Feel free to contact me for any inquiries or collaborations.</p>
</body>
</html>
```

In this example, we have used the `
` tag to insert line breaks within paragraphs. Each time the `
` tag is used, it creates a new line, effectively breaking the content onto a new line.

For instance, in the first paragraph under the "Welcome to My Website" section, we have used the `
` tag to separate the text into two lines. Similarly, in the "About Me" and "Contact Information" sections, line breaks are introduced to create separate lines of text within each paragraph.

Using the `
` tag is particularly useful when you want to force a line break without starting a new paragraph. It is commonly used for addresses, song lyrics, poetry, or other instances where a line break is required without the need for creating a new paragraph.

Remember that the `
` tag doesn't require a closing tag since it is a self-closing tag. You can use it directly within the content, and it will create a line break at that point.

However, it's generally recommended to use line breaks sparingly and maintain proper HTML structure by using appropriate elements like paragraphs or headings for better semantic markup and maintainability of the code.

HORIZONTAL LINES

In HTML, you can create horizontal lines using the `<hr>` element. The `<hr>` tag represents a thematic break or a horizontal rule in an HTML document. It is commonly used to separate sections of content.

Here's an example of how to use the `<hr>` tag in HTML :-

```
html
Copy code

<!DOCTYPE html>
<html>
<head>
  <title>Horizontal Line Example</title>
</head>
<body>
  <h1>Heading 1</h1>
  <p>This is some text.</p>

  <hr>

  <h2>Heading 2</h2>
  <p>This is more text.</p>
</body>
</html>
```

In the above example, the `<hr>` tag is placed between the first and second sections of content. When the HTML document is rendered in a web browser, it will display a horizontal line separating the two sections.

PRESERVE FORMATTING

Preserving formatting in HTML means maintaining the original appearance of text, including spaces, line breaks, and other formatting elements, when displaying it on a webpage. This is especially useful when you want to display code snippets, poetry, or any content that relies on specific formatting.

To preserve formatting in HTML, you can use the `<pre>` (preformatted) element. The `<pre>` tag tells the browser to display the enclosed text exactly as it appears in the HTML code, preserving spaces and line breaks.

```
html
Copy code

<!DOCTYPE html>
<html>
<head>
  <title>Preserve Formatting Example</title>
</head>
<body>
  <pre>
    This is some text
    with preserved
    formatting.
  </pre>
</body>
</html>
```

In the above example, the text within the `<pre>` tags will be displayed on the webpage exactly as it appears in the HTML code. The extra spaces and line breaks will be maintained.

```
vbnet
Copy code

This is some text
with preserved
formatting.
```

Using the `<pre>` element is particularly useful when displaying code snippets, as it allows you to maintain the indentation and spacing of the code for readability.

```
html
Copy code

<!DOCTYPE html>
<html>
<head>
  <title>Code Snippet Example</title>
</head>
<body>
  <pre>
    <code>
      function helloWorld() {
        console.log("Hello, world!");
      }
      helloWorld();
    </code>
  </pre>
</body>
</html>
```

In this example, the `<pre>` element is combined with the `<code>` element to preserve the formatting of the code snippet. The code will be displayed on the webpage with proper indentation and line breaks :-

```
javascript
Copy code

function helloWorld() {
  console.log("Hello, world!");
}
helloWorld();
```

HTML TAG VS ELEMENT

HTML Tag: An HTML tag is a special code that tells a web browser how to display content. Tags are written using angle brackets (< and >) and usually come in pairs, with an opening tag and a closing tag. The opening tag indicates the beginning of an element, and the closing tag marks the end of the element. Tags are used to structure and format content on a webpage.

Example of an HTML tag:

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

In the above example, `<p>` is the opening tag, and `</p>` is the closing tag. They enclose the paragraph content and indicate that it should be displayed as a paragraph.

HTML Element: An HTML element refers to a complete unit of content within an HTML document. It consists of an opening tag, the content, and a closing tag. Elements can be as simple as a single tag or more complex with nested elements inside.

Example of an HTML element:

`<h1>This is a heading</h1>`

In the above example, `<h1>` is the opening tag, `</h1>` is the closing tag, and "This is a heading" is the content of the heading element. The entire structure represents a heading on the webpage.

HTML elements can include text, images, links, lists, tables, forms, and more. They provide structure and define the various parts of a webpage, allowing the browser to understand how to render and display the content.

NESTED HTML ELEMENTS

Nested HTML elements are HTML elements that are placed inside other HTML elements. In other words, one element is contained within another element. This hierarchical structure allows for organizing and structuring content on a webpage.

When an HTML element is nested, it means that it is placed between the opening and closing tags of another element. This establishes a parent-child relationship between the elements, where the outer element is the parent and the inner element is the child.

Here's an example of nested HTML elements:

```
html


<h1>This is a heading</h1>
  <p>This is a paragraph.</p>
</div>


```

In the above example, the `<h1>` element and the `<p>` element are nested within the `<div>` element. The `<div>` element acts as the parent element, and the heading and paragraph elements are the child elements. The content inside the child elements is displayed within the context of the parent element.

Nested elements can be nested further to create deeper levels of hierarchy. For instance:

```
html


<h1>This is a heading</h1>
  <p>This is a paragraph.</p>
  <ul>
    <li>List item 1</li>
    <li>List item 2</li>
  </ul>
</div>


```

In this example, the `` element (unordered list) is nested within the same `<div>` element. The `` element serves as a child element of the `<div>` element, and the list items (``) are children of the `` element.

By nesting elements, you can create complex structures and layouts on a webpage. The parent-child relationship allows you to group related elements together and apply styling or formatting to them collectively.

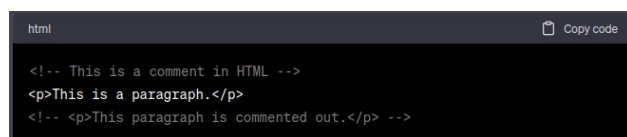
COMMENTS IN HTML

Comments in HTML are used to add notes or explanations within the HTML code that are not displayed on the webpage. They are useful for providing information to other developers or for

temporarily disabling a portion of the code without deleting it.

In HTML, comments are written using the `<!--` and `-->` delimiters. Anything between these delimiters is considered a comment and is ignored by the web browser when rendering the webpage.

Here's an example of an HTML comment:



```
html
Copy code

<!-- This is a comment in HTML -->
<p>This is a paragraph.</p>
<!-- <p>This paragraph is commented out.</p> -->
```

In the above example, the first line is an HTML comment that provides a note or explanation. The comment is not displayed on the webpage. The second line is a regular HTML element (a paragraph in this case). The third line is a commented-out paragraph, indicated by the `<!--` and `-->` delimiters. This paragraph is not rendered by the browser because it is within the comment.

HTML comments are useful for various purposes, such as:

1. **Documenting the code:** Developers can add comments to describe the purpose or functionality of a particular section of code, making it easier for others to understand.
2. **Debugging or troubleshooting:** Comments can be used to temporarily remove or disable a portion of the code to identify or isolate an issue.
3. **Leaving reminders or to-do lists:** Developers can use comments to leave reminders or notes for themselves or their team members about pending tasks or improvements.

It's important to note that comments should be used judiciously and kept up to date. Excessive or outdated comments can clutter the code and make it harder to maintain.

HTML ATTRIBUTES



```
html
Copy code

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

In the above example, the `<a>` element represents a hyperlink, and the `href` attribute is used to specify the destination URL that the hyperlink should point to. The value of the `href` attribute is set to "<https://www.example.com>". When a user clicks on the link, it will navigate to the specified URL.

HTML attributes can be used to achieve various effects, such as:

1. **Changing element behavior:** Attributes like `href`, `src`, `disabled`, and `required` modify how elements like links, images, buttons, and form inputs function.
2. **Styling elements:** Attributes like `class` and `style` are used to apply CSS classes or inline styles to elements, allowing for custom styling and presentation.
3. **Providing additional information:** Attributes like `alt`, `title`, and `data-*` attributes can provide supplementary information about elements for accessibility, tooltips, or custom data storage.

It's worth noting that not all HTML elements have attributes, and the available attributes may vary depending on the element. HTML attributes play a crucial role in extending the functionality and customization of HTML elements within web pages.

ITALIC TEXT

In HTML, you can make text appear in italics by using the `` element or the `<i>` element. Both elements are used to indicate emphasis or to italicize text.

Here's an example of using the `` element:

<p>This is italic text.</p>

In the above example, the word "italic" is enclosed within the tags. When the HTML document is rendered in a web browser, the text within the tags will appear in italics:

This is italic text.

Alternatively, you can use the <i> element to achieve the same result:

<p>This is <i>italic</i> text.</p>

Both and <i> elements are typically rendered in italics by default. However, it's important to note that their usage carries different semantic meanings:

- represents emphasized text, which typically implies importance or stress within the context of the content. Web browsers often render it as italicized text by default.
- <i> represents text in an alternative voice or mood, or denotes a technical term, a phrase in another language, or a book or magazine title. By default, web browsers render it as italicized text, but it does not necessarily imply emphasis or importance.

It is recommended to use the element when emphasizing text for semantic purposes. However, if the purpose is purely visual styling, either or <i> can be used to achieve italicized text.

UNDERLINED TEXT

To underline text means to add a line below the text to indicate emphasis or to differentiate it from the surrounding content. It is a formatting technique used to visually highlight or draw attention to specific words or phrases within a block of text.

In HTML, you can underline text using the <u> (underline) element or by applying CSS styles.

Using the <u> element:

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

In the above example, the <u> element is used to enclose the word "underlined". When rendered in a web browser, the text "underlined" will appear with a line underneath.

Using CSS styles:

<p style="text-decoration: underline;">This is underlined text.</p>

In this example, the style attribute is applied to the <p> element with the value text-decoration: underline;. This CSS property instructs the browser to underline the text within the <p> element.

Underlining text is a simple and commonly used way to add emphasis or indicate links within HTML content. However, keep in mind that underlining is often associated with hyperlinks, so it's generally recommended to reserve underlining for links to avoid confusing users. For non-link text emphasis, alternative styling techniques like bolding or color changes can be considered.

SUPERSCRIPT TEXT

Subscript text in HTML refers to text that appears below the baseline of the surrounding text, typically used for representing subscripts or mathematical formulas. It is commonly used for displaying chemical formulas, mathematical equations, or footnotes.

In HTML, you can create subscript text using the <sub> (subscript) element. The <sub> tag is an inline-level element that indicates that the enclosed text should be displayed in a subscript style.

Here's an example of using the <sub> element in HTML:

<p>H₂O is the chemical formula for water.</p>

In the above example, the <sub> element is used to enclose the "2" in the chemical formula

for water (H₂O). When rendered in a web browser, the "2" will appear as subscript text, appearing below the baseline of the surrounding text.

It's important to note that the actual rendering and appearance of subscript text may vary depending on the browser and the CSS styles applied to the element. However, the `<sub>` element is specifically designed to convey the intent of displaying text as a subscript.

Additionally, you can also apply CSS styles to achieve subscript-like effects using properties like `vertical-align` or by using custom classes. However, using the `<sub>` element is the recommended and semantically correct way to represent subscript text in HTML.

SUBSCRIPT TEXT

Superscript text in HTML refers to text that appears above the baseline of the surrounding text, typically used for representing superscripts, mathematical exponents, or ordinal indicators. It is commonly used for displaying mathematical formulas, footnotes, or other notations.

In HTML, you can create superscript text using the `<sup>` (superscript) element. The `<sup>` tag is an inline-level element that indicates that the enclosed text should be displayed in a superscript style.

Here's an example of using the `<sup>` element in HTML :-

`<p>E=mc2</sup> is the famous equation by Albert Einstein.</p>`

In the above example, the `<sup>` element is used to enclose the "2" in the equation $E=mc^2$. When rendered in a web browser, the "2" will appear as superscript text, positioned above the baseline of the surrounding text.

Similarly to subscript text, the actual rendering and appearance of superscript text may vary depending on the browser and the CSS styles applied to the element. However, using the `<sup>` element is the recommended and

semantically correct way to represent superscript text in HTML.

It's worth noting that you can also apply CSS styles to achieve superscript-like effects using properties like `vertical-align` or by using custom classes. However, using the `<sup>` element ensures clarity and accessibility for assistive technologies and future-proofing the content structure.

LARGER TEXT

In HTML, you can make text appear larger by using the `<h1>` to `<h6>` elements or by applying CSS styles to increase the font size. Both methods allow you to adjust the size of the text to create visual emphasis or hierarchy within your HTML content.

Using heading elements (`<h1>` to `<h6>`):

`<h1>This is the largest heading</h1>`
`<h2>This is a smaller heading</h2>`

In the above example, the `<h1>` element represents the largest heading size, while the `<h2>` element represents a smaller heading size. The browser will render these headings with predefined sizes, where `<h1>` is the largest and `<h6>` is the smallest.

SMALLER TEXT

In HTML, you can make text appear smaller by using the `<small>` element or by applying CSS styles to decrease the font size. Both methods allow you to adjust the size of the text to create visual emphasis or to indicate secondary information within your HTML content.

Using the `<small>` element:

`<small>This is smaller text.</small>`

STRONG TEXT

In HTML, the `` element is used to indicate that the enclosed text should be displayed as strong or important content. By default, the text within the `` element is

rendered in bold to visually emphasize its significance.

Here's an example of using the `` element in HTML:

```
<p>This is a <strong>strong</strong> statement.</p>
```

In the above example, the word "strong" is enclosed within the `` tags. When rendered in a web browser, the word "strong" will appear in bold, indicating its importance or emphasis within the sentence.

The `` element provides semantic meaning to the text, indicating to browsers, search engines, and assistive technologies that the enclosed content is of particular importance or relevance. It is typically used to highlight keywords, important phrases, or critical information within the content.

It's worth noting that the visual representation of strong text (e.g., bold) can be modified using CSS styles. However, the `` element should be used to convey the semantic meaning of strong content, while the actual styling can be adjusted using CSS.

HTML META TAGS

HTML meta tags provide information about an HTML document and its characteristics. They are placed within the `<head>` section of an HTML document and are not displayed on the webpage itself. Meta tags are important for search engines, social media platforms, and browsers to understand and process the content of a webpage correctly.

Here are some commonly used HTML meta tags:

1. `<meta charset="UTF-8">`: Specifies the character encoding for the HTML document. It ensures that the browser displays the text correctly.
2. `<meta name="viewport" content="width=device-width, initial-scale=1.0">`: Sets the viewport properties

for responsive design and mobile devices. It ensures that the webpage adapts to different screen sizes and provides a consistent user experience.

3. `<meta name="description" content="A brief description of the webpage">`: Provides a concise summary or description of the webpage's content. This information is often displayed by search engines in search results.
4. `<meta name="keywords" content="keyword1, keyword2, keyword3">`: Specifies the keywords or phrases that describe the content of the webpage. While less important for search engine rankings nowadays, they can still provide context to search engines.
5. `<meta name="author" content="Author Name">`: Indicates the author of the webpage.
6. `<meta name="robots" content="index, follow">`: Instructs search engine robots on how to crawl and index the webpage. The values can include "index" (to include the webpage in search results) and "follow" (to follow and index links on the page).
7. `<meta http-equiv="refresh" content="5;url=https://example.com/">`: Automatically redirects the browser to another URL after a specified time (5 seconds in this example).

These are just a few examples of HTML meta tags. There are other meta tags available for specific purposes, such as Open Graph tags for social media sharing, Twitter cards, or specifying the language of the document. The choice and use of meta tags depend on the specific requirements and goals of your webpage.

HTML LISTS

In simpler terms, HTML lists are used to organize and present information in a structured manner. Lists allow you to group related items or content together and provide a clear visual hierarchy. There are three types of HTML lists: unordered lists, ordered lists, and definition lists.

HTML UNORDERED LISTS

In simpler terms, an unordered list in HTML is a way to present a collection of items without any specific order or sequence. It is often represented as a bulleted list, where each item is preceded by a bullet point.

To create an unordered list in HTML, you use the `` element along with the `` element for each list item.

Here's an example of an unordered list in HTML:

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

In the above example, the `` element represents the unordered list, and each list item is enclosed within an `` element. The browser will render this as a bulleted list:

- Item 1
- Item 2
- Item 3

Unordered lists are useful when the order of the items is not important, and you want to present them as a simple list without any particular sequence. They are commonly used for features, options, bullet points, or any situation where the items don't need to be numbered or ranked.

You can customize the appearance of the bullets using CSS to change their shape, size, color, or even remove them completely. The unordered list structure (`` and ``) remains the same, and the styling can be adjusted to match the design and style of your webpage.

Unordered lists (``) in HTML support various attributes to modify their behavior and appearance. Let's explore some commonly used attributes:

type: This attribute specifies the type of bullet or marker used in the list. The default value is "disc", which displays a filled circle. Other common values include "square" (square bullet) and "circle" (hollow circle).

Example 1: Using "circle" type attribute:

```
html
<ul type="circle">
  <li>Apple</li>
  <li>Orange</li>
  <li>Banana</li>
</ul>
```

Example 2: Using "disc" type attribute (default):

```
html
<ul type="disc">
  <li>Red</li>
  <li>Green</li>
  <li>Blue</li>
</ul>
```

Example 3: Using "square" type attribute:

```
html
<ul type="square">
  <li>Cat</li>
  <li>Dog</li>
  <li>Rabbit</li>
</ul>
```

HTML ORDERED LISTS

In simpler terms, an ordered list in HTML is a way to present a collection of items in a specific sequence or order. Each item in the list is automatically numbered or ordered. It is commonly represented as a numbered list.

To create an ordered list in HTML, you use the `` element along with the `` element for each list item.

Here's an example of an ordered list in HTML:

```
html
<ol>
  <li>First item</li>
  <li>Second item</li>
  <li>Third item</li>
</ol>
```

In the above example, the `` element represents the ordered list, and each list item is enclosed within an `` element. The browser will render this as a numbered list:

1. First item
2. Second item
3. Third item

Ordered lists are useful when you want to present a series of items in a specific order or sequence. The numbering or ordering of the items provides a clear structure and indicates the relative importance or flow of information.

Similar to unordered lists, you can customize the appearance of the numbering using CSS to change the style, format, or alignment. The ordered list structure (`` and ``) remains the same, and the styling can be adjusted to match the design and style of your webpage.

In summary, an ordered list in HTML is a way to display a list of items in a specific sequence with automatic numbering. It helps to organize information and maintain a logical order for the content being presented.

The type Attribute

You can use **type** attribute for `` tag to specify the type of numbering you like. By default it is a number. Following are the possible options:

```
<ol type="1"> - Default-Case Numerals.  
<ol type="I"> - Upper-Case Numerals.  
<ol type="i"> - Lower-Case Numerals.  
<ol type="a"> - Lower-Case Letters.  
<ol type="A"> - Upper-Case Letters.
```

The start Attribute

You can use **start** attribute for `` tag to specify the starting point of numbering you need. Following are the possible options:

```
<ol type="1" start="4"> - Numerals starts with 4.  
<ol type="I" start="4"> - Numerals starts with IV.  
<ol type="i" start="4"> - Numerals starts with iv.  
<ol type="a" start="4"> - Letters starts with d.  
<ol type="A" start="4"> - Letters starts with D.
```

DATA DEFINITION LISTS (DDL)

In HTML, a definition list (`<dl>`) is a way to present a list of terms and their corresponding definitions. It is often used when you need to provide a clear and structured explanation of concepts, glossary terms, or any other set of terms and their meanings.

A definition list consists of two main components: term and definition.

To create a definition list in HTML, you use the `<dl>` element along with the `<dt>` element for terms and the `<dd>` element for definitions.

Here's an example of a definition list in HTML:

```
html Copy code  
  
<dl>  
  <dt>Term 1</dt>  
  <dd>Definition 1</dd>  
  <dt>Term 2</dt>  
  <dd>Definition 2</dd>  
</dl>
```

In the above example, the `<dl>` element represents the definition list. Each term is enclosed within the `<dt>` element, and each definition is enclosed within the `<dd>` element. The browser will render this as a structured list:

Term 1 Definition 1

Term 2 Definition 2

The `<dt>` element is used to mark the beginning of a term or item in the list, while the `<dd>` element represents the corresponding definition or explanation. The terms and their definitions are visually associated with each other, creating a clear and organized presentation of information.

Definition lists can be used for various purposes, such as glossaries, FAQs, or any situation where you need to pair terms with their meanings. The structure of the definition list allows for easy readability and comprehension of the content.

It's worth noting that you can apply CSS styles to further customize the appearance of definition lists, including adjusting spacing, font styles, or alignment. However, the fundamental structure of the definition list (`<dl>`, `<dt>`, and `<dd>`) remains the same, and the styling can be adjusted to match the design and style of your webpage.

INSERT IMAGE

HTML images are used to display visual content, such as photographs, illustrations, icons, or logos, on a webpage. Images are an essential part of web design and can enhance the visual appeal and engagement of a webpage.

To insert an image in HTML, you use the `` element and specify the source (URL) of the image using the `src` attribute. Here's an example:

```

```

In the above example, the `src` attribute specifies the URL or file path of the image file, and the `alt` attribute provides alternative text that describes the image. The `alt` attribute is important for accessibility, as it is displayed by assistive technologies and search engines when the image cannot be loaded or is unavailable.

In addition to the `src` and `alt` attributes, HTML images support various other attributes that can modify their behavior and appearance. Some commonly used attributes for images include :-

- **width and height:** These attributes specify the width and height dimensions of the image in pixels. You can set these attributes to control the display size of the image on the webpage.
- **title:** The title attribute provides a tooltip or additional information about the image when a user hovers over it with the cursor. It is displayed as a tooltip text.
- **class and id:** These attributes allow you to assign CSS classes or unique identifiers to the image element. This allows you to apply custom styles or manipulate the image using CSS or JavaScript.
- **style:** The style attribute lets you apply inline CSS styles to the image element. You can use it to adjust properties like borders, margins, padding, or positioning.

These are just a few examples of attributes that can be used with HTML images. Depending on your requirements, you may also utilize attributes such as `srcset` for responsive images, `loading` to control image loading behavior, or `usemap` to associate an image with an image map for clickable regions.

Remember, when using images on your webpage, it's essential to consider factors such as image

size optimization, responsive design, and alternative text for accessibility to ensure a positive user experience.

HTML TEXT LINKS

In simpler terms, HTML text links are clickable elements on a webpage that allow users to navigate to other web pages or specific parts of a page. These links are commonly known as hyperlinks.

To create a text link in HTML, you use the `<a>` (anchor) tag. The content placed between the opening `<a>` tag and the closing `` tag becomes clickable, and when a user clicks on that part, they are directed to the linked document or webpage.

Here's a simplified explanation:

- Hyperlinks are used to connect different web pages or specific parts of a page.
- You can create hyperlinks using text or images on a webpage.
- To create a text link, you use the `<a>` (anchor) tag in HTML.
- Anything placed between the opening and closing `<a>` tags becomes clickable.
- When a user clicks on the clickable part, they are taken to the linked document or webpage.

For example, if you want to create a link to a webpage called "example.html" and display the text "Click here" as the clickable part, you can use the following HTML code :-

```
<a href="example.html">Click here</a>
```

THE TARGET ATTRIBUTE

In HTML, the `target` attribute is used to specify where a linked document or resource should be opened when a user clicks on a hyperlink. It determines the target browsing context or window in which the linked content will be displayed.

The target attribute can take several values :-

- `_blank`: Opens the linked document or resource in a new browser tab or window.
- `_self`: Opens the linked document or resource in the same browsing context or window (the default behavior if the target attribute is not specified).
- `_parent`: Opens the linked document or resource in the parent frame or window, if the current page is inside a frameset.
- `_top`: Opens the linked document or resource in the top-level browsing context, discarding any frames if the current page is inside a frameset.
- A custom frame or window name: If you specify a custom name (e.g., `target="myframe"`), it opens the linked document or resource in a specific named frame or window.

Here's an example that demonstrates the usage of the target attribute:

```
<a href="example.html" target="_blank">Open in new window</a>
```

In this example, the linked document "example.html" will be opened in a new browser tab or window when the user clicks on the link, as the target attribute is set to `_blank`.

It's worth noting that the behavior of the target attribute can be influenced by the user's browser settings, such as whether they have enabled pop-up blockers or overridden the default behavior. Additionally, it's important to use the target attribute responsibly and consider the impact on the user experience and accessibility.

When using the target attribute, it is recommended to provide clear indications to users that a link will open in a new window or tab to avoid any confusion or unexpected behaviors.

LINKING TO A PAGE SECTION

To create a link to a specific section within a webpage, you can use the name attribute in HTML. This process involves two steps:

Step 1: Creating a named anchor: To designate the location within the webpage where you want to link to, you first need to create a named anchor. This is done by using the `<a>` tag with the name attribute.

```
<a name="top"></a>
```

In this example, the `<a>` tag with the name attribute is placed at the top of the desired section. The value "top" is assigned to the name attribute, but you can choose any name you prefer.

Step 2: Creating a hyperlink to the named anchor: After creating the named anchor, you can create a hyperlink that will take the user to that specific section. To do this, you use the `<a>` tag again, but with the href attribute containing the `"#"` symbol followed by the name of the anchor.

```
<a href="#top">Go to the Top</a>
```

In this example, the `<a>` tag with the href attribute is used. The value `"#top"` in the href attribute specifies that the link should take the user to the section with the "top" named anchor.

When the link is rendered in a web browser, it will display as "Go to the Top." Clicking on this link will scroll the page to the section where the named anchor is placed (in this case, the top of the page).

This technique allows you to create internal page navigation within a webpage, enabling users to jump directly to specific sections of the content. It is particularly useful for long web pages or articles where users may want to navigate easily to different parts of the page.

HTML IMAGE LINKS

Here's a breakdown and explanation of the code:

```
<a href="page1.html"></a>
```

: This part creates the hyperlink and specifies the destination page or URL that the user will be directed to when they click on the image. In this example, "page1.html"

is used as the destination, but you can replace it with the desired URL or page name.

****: This part inserts the image within the hyperlink. The src attribute specifies the image source or file path. In this case, "aa.jpg" is used as the image source, but you should replace it with the actual source file of your image.

When this code is rendered in a web browser, it will display In this example, the `<table>` element represents the table. Each row is defined using the `<tr>` element, and within each row, cells are created using the `<td>` element. The browser will render this code as a simple table with two rows and two columns:an image that is clickable. Clicking on the image will take the user to the specified destination page (in this case, "page1.html").

Here's an example with placeholders:

HTML TABLES

HTML tables are used to organize and display tabular data in a structured grid format. They provide a way to present data in rows and columns, allowing for easy comprehension and comparison of information. Tables consist of three main components: table, table rows, and table cells.

To create a table in HTML, you use the `<table>` element. Within the `<table>` element, you define rows using the `<tr>` element, and each cell within a row is represented by the `<td>` element.

Here's an example of a simple HTML table with two rows and two columns:

```
html
Copy code

<table>
  <tr>
    <td>Cell 1,1</td>
    <td>Cell 1,2</td>
  </tr>
  <tr>
    <td>Cell 2,1</td>
    <td>Cell 2,2</td>
  </tr>
</table>
```

In this example, the `<table>` element represents the table. Each row is defined using the `<tr>` element, and within each row, cells are created using the `<td>` element. The browser will render this code as a simple table with two rows and two columns:

Tables can contain more complex structures, including header cells (`<th>`) for column headings, spanning cells across multiple rows or columns, or nested tables for advanced layouts. CSS styles can also be applied to tables to control their appearance, such as borders, backgrounds, and alignment.

HTML tables are versatile and widely used for presenting data in various contexts, such as financial reports, schedules, pricing information, and more. They provide an effective way to organize and present tabular data in a structured and accessible manner

Cell 1,1 Cell 1,2

Cell 2,1 Cell 2,2

COLSPAN AND ROWSPAN ATTRIBUTES

The colspan and rowspan attributes in HTML are used to control the spanning or merging of cells across multiple columns or rows within a table. They allow you to create cells that span across multiple columns or rows, effectively merging them into a larger cell.

- 1. colspan attribute:** The colspan attribute specifies the number of columns a cell should span or occupy horizontally.

```
html
Copy code

<table>
  <tr>
    <td>Cell 1</td>
    <td colspan="2">Cell 2 spans 2 columns</td>
  </tr>
  <tr>
    <td colspan="3">Cell 3 spans 3 columns</td>
  </tr>
</table>
```

In the above example, the first row contains three cells. The second cell (Cell 2) has a colspan attribute set to 2, which means it spans two columns. The third cell (Cell 3) has a

colspan attribute of 3, spanning all three columns. The resulting table will look like this:

Cell 1 **Cell 2 spans 2 columns**
Cell 3 spans 3 columns

Rowspan attribute: The rowspan attribute specifies the number of rows a cell should span or occupy vertically.

Example:

```
html
Copy code

<!DOCTYPE html>
<html>
<head>
  <title>Table with rowspan Example</title>
</head>
<body>
  <table border="1">
    <tr>
      <th>Header 1</th>
      <th>Header 2</th>
      <th>Header 3</th>
    </tr>
    <tr>
      <td rowspan="2">Cell 1</td>
      <td>Cell 2</td>
      <td>Cell 3</td>
    </tr>
    <tr>
      <td>Cell 4</td>
      <td>Cell 5</td>
    </tr>
    <tr>
      <td>Cell 6</td>
      <td>Cell 7</td>
      <td>Cell 8</td>
    </tr>
  </table>
</body>
</html>
```

In this example, we have a simple table with three columns and three rows. The cell in the first column (Cell 1) spans two rows vertically using the rowspan="2" attribute. This means that the cell occupies the space of two rows, merging with the cells below it.

Here's a representation of the table structure:

```
mathematica
Copy code

+-----+-----+-----+
| Header 1 | Header 2 | Header 3 |
+-----+-----+-----+
| Cell 1   | Cell 2   | Cell 3   |
+-----+-----+-----+
|          | Cell 4   | Cell 5   |
| Cell 1   | Cell 6   | Cell 7   |
+-----+-----+-----+
|          | Cell 8   |          |
+-----+-----+-----+
```

As shown in the table structure above, the Cell 1 spans two rows, merging with the cells Cell 4 and Cell 6 below it.

The rowspan attribute is useful when you want to create cells that span multiple rows, such as when you want to create table headers or cells that encompass larger sections of data.

Note that the table structure is represented with simple ASCII characters in the example above. When rendered in a browser, the table will be displayed with borders around the cells due to the border="1" attribute set on the <table> tag.

TABLE BACKGROUNDS

Using the bgcolor attribute: This attribute allows you to set the background color for the entire table or for individual cells within the table. It provides a way to add a solid color as the background.

```
html
Copy code

<table bgcolor="#e6e6e6">
  <tr>
    <td bgcolor="#ffcccc">Cell 1</td>
    <td bgcolor="#ccffcc">Cell 2</td>
  </tr>
  <tr>
    <td bgcolor="#ccccff">Cell 3</td>
    <td bgcolor="#ffffcc">Cell 4</td>
  </tr>
</table>
```

In the example above, the bgcolor attribute is used to set the background color for the entire table to #e6e6e6, which is a light gray color. Additionally, each individual cell is given its own background color using the bgcolor attribute.

Using the background attribute :- This attribute allows you to set a background image for the entire table or for individual cells within the table. You can use an image file as the background, providing more visual customization.

```
html
Copy code

<table background="table-background.jpg">
  <tr>
    <td>Cell 1</td>
    <td>Cell 2</td>
  </tr>
  <tr>
    <td>Cell 3</td>
    <td>Cell 4</td>
  </tr>
</table>
```

In the above example, the background attribute is used to set an image file called table-background.jpg as the background for the entire table. The image will be repeated across the table if it doesn't cover the entire table area.

To set the border color for the table, you can use the bordercolor attribute. This attribute allows you to specify the color of the table border.

```
html
Copy code
<table border="1" bordercolor="#ff0000">
  <tr>
    <td>Cell 1</td>
    <td>Cell 2</td>
  </tr>
  <tr>
    <td>Cell 3</td>
    <td>Cell 4</td>
  </tr>
</table>
```

In this example, the bordercolor attribute is used to set the border color of the table to #ff0000, which is red.

TABLE WIDTH AND HEIGHT

Table width and height in HTML refer to the dimensions of a table element. The width determines the horizontal space occupied by the table, while the height determines the vertical space.

Here's a simpler example that demonstrates setting the width and height of a table:

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
  <title>Table Width and Height Example</title>
</head>
<body>
  <table width="400" height="200">
    <tr>
      <td>Cell 1</td>
      <td>Cell 2</td>
    </tr>
    <tr>
      <td>Cell 3</td>
      <td>Cell 4</td>
    </tr>
  </table>
</body>
</html>
```

In this example, the <table> tag has the width attribute set to "400" and the height attribute set to "200". These values determine the width and

height of the table, respectively. You can adjust these values to fit your desired dimensions.

The table will occupy a horizontal space of 400 pixels and a vertical space of 200 pixels. The cells within the table will adjust their size accordingly to fit within the specified width and height.

Note that the width and height attributes are optional, and if not provided, the table will adjust its dimensions based on the content it contains. However, specifying width and height can provide more control over the layout and appearance of the table.

It's important to mention that setting fixed table dimensions using width and height attributes may affect the responsiveness of the table on different screen sizes. For responsive designs, it is often recommended to use CSS and relative units like percentages or viewport units for width and height, which allows the table to adapt to different screen sizes.

HTML FRAMES

HTML frames allow you to divide your web browser window into multiple sections, where each section can display a separate HTML document. The collection of frames in a browser window is called a frameset. Similar to how tables organize content into rows and columns, frames divide the window into rows and columns.

To create frames in HTML, instead of using the <body> tag, you use the <frameset> tag. The <frameset> tag defines how the window should be divided into frames. The rows attribute of the <frameset> tag defines horizontal frames (rows), and the cols attribute defines vertical frames (columns). Each frame is indicated by the <frame> tag, which specifies the HTML document to be displayed within that frame.

Here's an example of creating frames using HTML

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
  <title>Frames Example</title>
</head>
<frameset rows="50%, 50%">
  <frame src="frame1.html">
  <frame src="frame2.html">
</frameset>
</html>
```

In this example, we have a frameset with two rows, each occupying 50% of the window's height. The `<frame>` tags specify the HTML documents (frame1.html and frame2.html) to be loaded into each frame. Each frame will display its respective HTML document within the divided window.

THE FRAMESET TAG ATTRIBUTES

- **cols:** Specifies the number of columns in the frameset and the size of each column.
- **rows:** Works similar to cols but specifies the rows in the frameset.
- **border:** Specifies the width of the border around each frame. A value of zero means no border.

<frame> Tag Attributes:

- **src:** Specifies the file name or URL of the content to be loaded in the frame.
- **name:** Allows you to assign a name to the frame, indicating where a document should be loaded.
- **frameborder:** Controls the display of borders around the frame. Use `frameborder="0"` to hide the border.
- **noresize:** Prevents users from resizing the frame.
- **scrolling:** Controls the appearance of scrollbars within the frame. Options are "yes", "no", or "auto".

The example mentions using frames to create a navigation bar in one frame and loading main pages into a separate frame. It demonstrates creating two columns within the frameset, where the first frame, 200 pixels wide, contains a

navigation menu (menu.htm), and the second column fills the remaining space with the main content (main.htm).

The target attribute is not explicitly mentioned in the given text, but it is commonly used in conjunction with frames. By specifying the target attribute in links within the navigation frame, you can define which frame the linked content should load into. In the example, the target frame is named "main_page," indicating that when a link in the navigation menu is clicked, the linked content will open in the "main" frame.

Frames allow for the separation of content into different sections, such as a persistent navigation bar, and dynamic loading of content in another frame. This approach enables simultaneous browsing and interaction with different parts of a webpage.

However, it's important to note that the use of frames has diminished over time due to its drawbacks and the availability of more modern techniques for achieving similar results, such as using CSS for layout or JavaScript frameworks for dynamic content.

Example :-

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
  <title>Frames Example</title>
</head>
<frameset cols="200, *">
  <frame src="menu.htm" name="navigation" frameborder="0" scrolling="no">
  <frame src="main.htm" name="content" frameborder="0">
</frameset>
</html>
```

In this example, a frameset is created with two columns specified by `cols="200, *"`. The first column has a width of 200 pixels, while the second column takes up the remaining space.

The frames within the frameset are defined using the `<frame>` tag. The first frame has its content loaded from menu.htm using the `src` attribute. It is assigned the name "navigation" using the `name` attribute. The `frameborder` attribute is set to "0" to hide the frame border, and `scrolling` is set to "no" to remove scrollbars.

The second frame loads its content from main.html and is assigned the name "content". Similar to the first frame, the frameborder attribute is set to "0" and there are no scrollbars.

This example demonstrates the usage of attributes such as cols, src, name, frameborder, and scrolling within a frameset. It separates the navigation menu in one frame and the main content in another frame, allowing for independent scrolling and targeted loading of content.

Note that the actual content of menu.htm and main.htm will depend on the specific implementation and the desired functionality of the frameset.

HTML FORMS

HTML forms are used to collect data from site visitors. They are commonly used for tasks like user registration, where information such as name, email address, and credit card details are collected. The data submitted through a form is typically processed by a backend application, which performs the necessary actions based on the collected data.

To create an HTML form, you use the <form> tag. It has attributes that specify the backend script that will process the form data (action) and the method used to send the data to the backend (method).

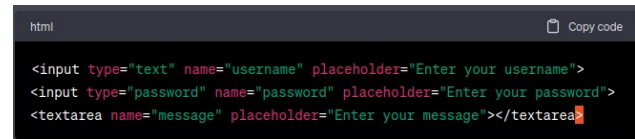
There are various types of form controls that can be used to collect data :-

Text Input Controls:

- **Single-line text input:** Used for collecting short inputs like names or search queries. Created using the <input> tag.
- **Password input:** Similar to single-line text input, but the characters entered are masked for security. Also created using the <input> tag.
- **Multi-line text input:** Used when the user needs to provide longer descriptions or

messages. Created using the <textarea> tag.

For single-line and password inputs, you use the <input> tag with the type attribute set to "text" or "password" respectively.



```
html
Copy code
<input type="text" name="username" placeholder="Enter your username">
<input type="password" name="password" placeholder="Enter your password">
<textarea name="message" placeholder="Enter your message"></textarea>
```

In the example above, the first two lines represent single-line text input and password input fields respectively, while the third line represents a multi-line text input field.

These input controls allow users to enter data into the form, such as their username, password, or longer messages, which can then be submitted to a backend application for processing.

Note: The name attribute is used to identify the input field when the form is submitted, and the placeholder attribute provides a hint or example of the expected input to the user.

HTML FORM CONTROLS

- Text Input Controls
- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Select boxes
- Hidden Controls
- Clickable Buttons
- Submit and Reset Button

Check Box Controls :- A checkbox control in HTML is used to present a binary choice to the user. It allows the user to select one or more options from a list of choices.

In simpler terms, a checkbox control is like a small box that can be checked or unchecked by the user. When checked, it indicates that a particular option is selected, and when unchecked, it means the option is not selected.

Here's a code example that demonstrates the usage of checkbox controls in HTML:

```
html
Copy code

<form>
  <input type="checkbox" id="option1" name="option1" value="Option 1" checked>
  <label for="option1">Option 1</label><br>

  <input type="checkbox" id="option2" name="option2" value="Option 2">
  <label for="option2">Option 2</label><br>

  <input type="checkbox" id="option3" name="option3" value="Option 3">
  <label for="option3">Option 3</label><br>
</form>
```

In this example, we have three checkbox controls. Each checkbox is created using the `<input>` tag with the `type` attribute set to "checkbox". The `id`, `name`, and `value` attributes are used to uniquely identify and handle the checkbox when the form is submitted.

The associated `<label>` elements provide a text label for each checkbox option. The `for` attribute of each label is set to the corresponding checkbox's `id`, which associates the label with the checkbox. Clicking on the label will also toggle the checkbox.

When the form is submitted, the selected checkbox options will be sent to the server-side application for further processing.

Remember that each checkbox should have a unique `id` and `name` to ensure proper identification and handling of the selected options.

Using checkbox controls in forms allows users to make multiple selections or indicate their preferences for various options, such as selecting multiple items from a shopping list or choosing preferences for notification settings.

If we want a checkbox to be checked by default when the page loads, you can use the `checked` attribute in the `<input>` tag.

Radio Box Controls

Attribute	Description
type	Indicates the type of input control and for checkbox input control it will be set to checkbox.
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
value	The value that will be used if the checkbox is selected.
checked	Set to <i>checked</i> if you want to select it by default.

A radio button control in HTML is used to present a set of mutually exclusive choices to the user. It allows the user to select only one option from a group of options.

Radio buttons are like small circular buttons, and when one radio button is selected, any previously selected radio button within the same group is automatically deselected.

Here's a definition and example of radio button controls, including some common attributes:

```
html
Copy code

<form>
  <input type="radio" id="option1" name="options" value="Option 1" checked>
  <label for="option1">Option 1</label><br>

  <input type="radio" id="option2" name="options" value="Option 2">
  <label for="option2">Option 2</label><br>

  <input type="radio" id="option3" name="options" value="Option 3">
  <label for="option3">Option 3</label><br>
</form>
```

In this example, we have a group of radio button controls. Each radio button is created using the `<input>` tag with the `type` attribute set to "radio". The `id`, `name`, and `value` attributes are used to uniquely identify and handle the radio buttons when the form is submitted.

The associated `<label>` elements provide a text label for each radio button option. The `for` attribute of each label is set to the corresponding radio button's `id`, which associates the label with the radio button. Clicking on the label will also select the corresponding radio button.

To ensure that one option is pre-selected when the page loads, you can add the `checked` attribute to one of the radio buttons within the group. In the example above, the first radio button (option1) has the `checked` attribute set, indicating that it is selected by default.

The `name` attribute is crucial for radio buttons as it groups them together. Only one radio button within the same group can be selected at a time. By giving them the same `name`, you ensure that only one option from the group can be chosen.

Radio buttons are commonly used when users need to select a single option from a predefined

list, such as choosing a gender or selecting a payment method.

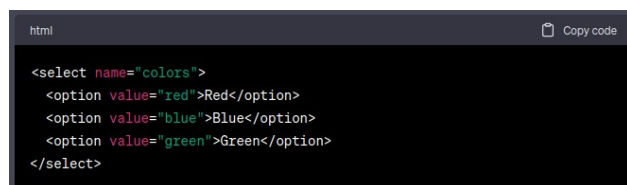
Feel free to modify the attributes like id, name, value, and checked to suit your specific needs and options.

SELECT BOX CONTROL

A select box control in HTML, also known as a dropdown menu, allows the user to choose one option from a list of predefined options. It presents a dropdown list that expands when clicked, displaying the available options for selection.

In simpler terms, a select box is like a menu that provides a selection of choices, and the user can choose one option from the list.

Here's an example of a select box control:



```
html
Copy code
<select name="colors">
  <option value="red">Red</option>
  <option value="blue">Blue</option>
  <option value="green">Green</option>
</select>
```

In this example, we have a select box created using the `<select>` tag. The name attribute is used to identify the select box when the form is submitted.

Within the `<select>` tag, we have multiple `<option>` tags. Each `<option>` tag represents an individual choice in the select box. The value attribute specifies the value associated with each option, which is sent to the server-side application when the form is submitted. The text within the `<option>` tags is the visible label for each choice.

When the page is rendered, the select box will display a default option. In this case, "Red" will be the initially displayed option. The user can click on the select box to expand the dropdown and choose a different option.

Select boxes are commonly used when there is a list of options from which the user needs to select one. For example, selecting a country, choosing a category, or picking a preferred time slot.

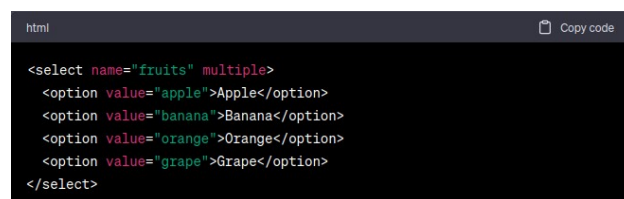
You can customize the select box further by adding attributes like **selected** to pre-select an option, **disabled** to disable the select box, or additional styling using CSS to modify its appearance.

SELECTING MULTIPLE VALUES

To select multiple values from a dropdown menu in HTML, you can use the `<select>` tag with the multiple attribute. This creates a select box that allows users to choose multiple options by holding down the Ctrl (or Command on macOS) key while selecting.

In simpler terms, it's like a dropdown menu where you can select more than one option by holding down a specific key.

Here's an example of a select box that allows multiple selections:



```
html
Copy code
<select name="fruits" multiple>
  <option value="apple">Apple</option>
  <option value="banana">Banana</option>
  <option value="orange">Orange</option>
  <option value="grape">Grape</option>
</select>
```

In this example, the `<select>` tag has the multiple attribute, which enables the selection of multiple options. The name attribute identifies the select box when the form is submitted.

The `<option>` tags within the `<select>` tag represent the individual choices in the dropdown. Each `<option>` tag has a value attribute that specifies the value associated with the option. When multiple options are selected, all the selected values are submitted as an array.

When the page is rendered, the select box will display a dropdown menu with the available options. By holding down the Ctrl (or Command) key and clicking on the options, the user can select multiple options.

The selected options can be accessed on the server-side when the form is submitted and processed.

Selecting multiple values in a dropdown is useful when you want to allow users to choose multiple

items from a list, such as selecting multiple interests, multiple ingredients, or multiple preferences.

Note: The appearance and behavior of multiple-select dropdowns may vary across different browsers and platforms. Additionally, consider providing clear instructions to users indicating that they can select multiple options by holding down the Ctrl (or Command) key.

FILE UPLOAD BOX

File Upload Box: A file upload box, also known as a file select box, allows users to select and upload files to a website. It is created using the `<input>` element with the type attribute set to "file".

In simpler terms, a file upload box is like a button that lets users choose a file from their computer and upload it to a website.

Example:

```
html
Copy code
<form>
  <input type="file" name="fileUpload">
</form>
```

In this example, we have an `<input>` tag with the type attribute set to "file". The name attribute is used to identify the file upload control when the form is submitted.

When the page is rendered, the file upload box will be displayed, allowing users to browse their computer and select a file to upload. Once the form is submitted, the selected file will be sent to the server-side application for processing.

Additional attributes can be used with the file upload box, such as the accept attribute, which specifies the types of files the server accepts. This attribute can be set to specific file extensions or MIME types to filter the files that can be selected.

CLICKABLE BUTTONS

In HTML, there are various ways to create clickable buttons. One way is to use the `<input>` tag and set its type attribute to "button". This allows you to create a button-like element that users can click.

```
html
Copy code
<input type="button" value="Click Me">
```

In this example, we have an `<input>` tag with the type attribute set to "button". The value attribute specifies the text displayed on the button.

When the page is rendered, a clickable button labeled "Click Me" will be displayed. You can add event handlers or JavaScript functions to handle actions when the button is clicked.

Clickable buttons are commonly used for form submissions, triggering specific actions, or navigating to different pages within a website.

HTML – EMBED MULTI MEDIA

Here's a simpler explanation of embedding multimedia in HTML using the `<embed>` tag and playing background audio using the `<bgsound>` tag:

Embedding Multimedia with `<embed>` Tag: The `<embed>` tag in HTML allows you to add multimedia, such as videos or sound, to your web page. It automatically provides browser controls for multimedia if the browser supports the `<embed>` tag and the specified media type.

In simpler terms, the `<embed>` tag is used to include videos or sound in your webpage, and if the browser supports it, it will display the multimedia content with built-in controls.

Example:

```
html
Copy code
<embed src="video.mp4" type="video/mp4" width="400" height="300">
```

In this example, we use the `<embed>` tag to embed a video file named "video.mp4" with the specified media type of "video/mp4". The width and height attributes define the dimensions of the video player on the webpage.

The browser will display the video and provide default controls for playing, pausing, and adjusting the playback.

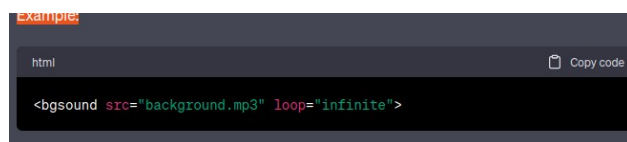
You can use different media types like Flash movies (.swf), AVI files (.avi), MOV files (.mov), or MPEG files (.mpeg) within the `<embed>` tag depending on the browser's support and the multimedia formats you want to embed.

Background Audio with `<bgsound>` Tag :-

The `<bgsound>` tag is used to play a soundtrack in the background of your webpage. However, it is supported only by Internet Explorer, and most other browsers ignore this tag.

In simpler terms, the `<bgsound>` tag allows you to play an audio file as background music on your webpage, primarily in Internet Explorer.

Example:



In this example, we use the `<bgsound>` tag to play an audio file named "background.mp3". The loop attribute is set to "infinite" to continuously replay the background sound whenever the user refreshes the browser.

Please note that `<bgsound>` is only supported by Internet Explorer and is not recognized by most other browsers. Therefore, for cross-browser compatibility, it is recommended to use alternative methods, such as the HTML5 `<audio>` tag with appropriate fallback options, to play background audio on your web pages.

CASCADING STYLE SHIT (CSS)

CSS (Cascading Style Sheets) is a language used to describe how HTML elements should look and be displayed on a web page. It allows you to control the appearance of various elements such as text, images, and layout.

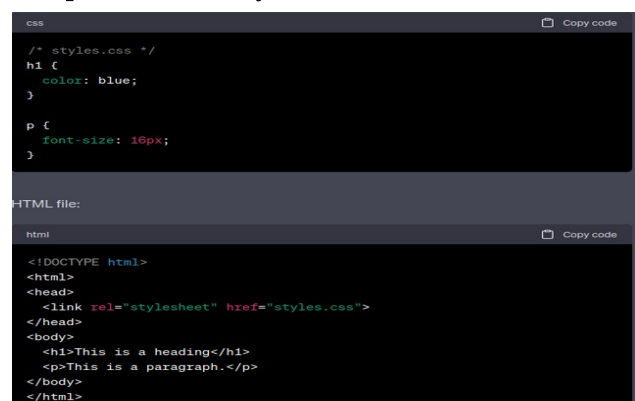
Instead of directly applying styles to HTML elements, CSS provides a way to separate the presentation (how things look) from the content and structure of a web page (what things are). This separation makes it easier to maintain and modify websites, especially when dealing with large-scale projects.

By using CSS, you can create a set of rules or styles that define the visual aspects of your website, such as the choice of fonts, colors, spacing, and positioning of elements. These styles can be saved in external CSS files, which can be applied to multiple web pages at once. This means that if you want to change the appearance of your entire website, you can simply replace the style sheet, and all the pages will adopt the new styles, making it easier to update the design or layout of your site.

THREE WAYS TO INSERT HTML IN TO A CSS DOCUMENT

External Style Sheet: An external style sheet is a separate CSS file that is linked to the HTML document using the `<link>` element. This method allows you to define styles in a separate file and then apply them to multiple HTML pages. It promotes re-usability and easier maintenance of styles.

Example: CSS file (styles.css):



Internal Style Sheet: An internal style sheet is defined within the <style> tags in the <head> section of an HTML document. This method allows you to apply styles specifically to that HTML page only.

Example:

```
html
css
Copy code

<!DOCTYPE html>
<html>
<head>
<style>
  h1 {
    color: blue;
  }

  p {
    font-size: 16px;
  }
</style>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Inline Style Sheet: Inline styles are directly applied to individual HTML elements using the style attribute. This method allows you to define styles inline with the HTML elements themselves. It is useful for applying styles to specific elements or overriding existing styles.

```
html
css
Copy code

<!DOCTYPE html>
<html>
<head>
</head>
<body>
  <h1 style="color: blue;">This is a heading</h1>
  <p style="font-size: 16px;">This is a paragraph.</p>
</body>
</html>
```

TYPES OF SELECTORS IN CSS

In CSS, selectors are used to target specific HTML elements or groups of elements that you want to apply styles to. There are various types of selectors available in CSS :-

1) Element Selectors

Element selectors in CSS are used to target specific HTML elements based on their tag name. They apply styles to all instances of that element throughout the document. Element selectors are

represented by the tag name without any additional identifiers.

```
css
Copy code

/* Targets all <h1> elements */
h1 {
  color: blue;
}
```

In the above example, the h1 element selector targets all <h1> elements and sets their color to blue. This means that every <h1> heading in the document will have the specified style applied.

Element selectors are simple and widely used because they allow you to apply styles to an entire category of elements. Here are a few more examples of element selectors :-

```
css
Copy code

/* Targets all <p> elements */
p {
  font-size: 16px;
}

/* Targets all <a> elements */
a {
  text-decoration: none;
}

/* Targets all <ul> elements */
ul {
  list-style-type: disc;
}
```

In the first example, the p selector targets all <p> elements and sets their font size to 16 pixels. This will affect all paragraphs on the page.

The second example targets all <a> elements (links) and removes the default underline by setting the text-decoration property to none.

The third example targets all elements (unordered lists) and changes their bullet style to a filled disc using the list-style-type property.

Element selectors are a fundamental part of CSS and are essential for applying styles to specific HTML elements throughout your web page.

2) CLASS SELECTORS

Class selectors in CSS are used to target HTML elements based on their assigned class attribute. The class attribute allows you to assign a specific name to one or multiple elements, and class

selectors are denoted by a dot (.) followed by the class name.

Class selectors provide a way to group and style elements that share the same class. This is especially useful when you want to apply the same styles to multiple elements without affecting other elements on the page.

```
css
/* Targets all elements with class "highlight" */
.highlight {
  background-color: yellow;
}
```

In the above example, the `.highlight` class selector targets all elements that have the `highlight` class. It sets their background color to yellow. To apply this style, you need to assign the class to the desired HTML elements using the class attribute.

```
HTML:
html
<p class="highlight">This paragraph has a yellow background.</p>
```

In this case, the paragraph element `<p>` has the `highlight` class assigned to it. As a result, the background color of the paragraph will be yellow.

You can apply class selectors to various HTML elements:

```
css
/* Targets all elements with class "button" */
.button {
  background-color: blue;
  color: white;
  padding: 10px 20px;
  border: none;
}

/* Targets all elements with class "section" */
.section {
  margin-bottom: 20px;
  padding: 10px;
  border: 1px solid gray;
}
```

In this example, the first class selector targets all elements with the class `button` and applies a set of styles to create a styled button. The second class selector targets elements with the class `section` and applies styles to create a styled section with a border and margin.

Class selectors offer a powerful way to apply consistent styles to multiple elements across your web page by grouping them based on common attributes. They provide flexibility and re-usability, allowing you to create more maintainable and organized CSS code.

```
html
<button class="button">Click me</button>
<div class="section">
  <h2>Section Title</h2>
  <p>This is a section content.</p>
</div>
```

3) ID SELECTORS

In CSS, an ID selector is used to target a specific HTML element based on its unique ID attribute. The ID attribute provides a unique identifier for an element within an HTML document. ID selectors are denoted by a hash symbol (`#`) followed by the ID value.

ID selectors are unique to each element and should only be used once per page. They are particularly useful when you want to apply specific styles to a single element or when you need to target a specific element using JavaScript.

Example :-

```
css
/* Targets the element with ID "logo" */
#logo {
  width: 200px;
  height: 150px;
}
```

In the above example, the `#logo` ID selector targets the HTML element with the ID attribute set to `"logo"`. It applies styles to that specific element, setting its width to 200 pixels and height to 150 pixels.

To use an ID selector, you need to assign a unique ID to the desired HTML element using the `id` attribute :-

```
html

```

In this HTML example, the `` element has the ID attribute set to "logo". As a result, the styles defined by the `#logo` ID selector in the CSS will be applied specifically to this image element.

It's important to note that ID selectors are intended for unique identification and should not be reused for multiple elements on the same page. An ID should be unique within the entire document.

Here's another example of an ID selector :-

```
css
/* Targets the element with ID "main-heading" */
#main-heading {
  color: red;
  font-size: 24px;
}

HTML:
<h1 id="main-heading">Welcome to our website!</h1>
```

In this case, the `#main-heading` ID selector targets the `<h1>` element with the ID "main-heading" and applies specific styles such as changing the text color to red and setting the font size to 24 pixels.

ID selectors are powerful tools to target specific elements within an HTML document. They provide a way to apply unique styles or behavior to individual elements, making them particularly useful in cases where you need to differentiate and style elements based on their unique characteristics.

GROUP SELECTORS

In CSS, group selectors allow you to apply the same styles to multiple selectors at once. Grouping selectors is a way to simplify your CSS code and avoid repeating the same styles for different selectors. To group selectors, you simply separate them with a comma (,).

Here's a breakdown of different types of group selectors:

1. **Element Group Selectors:** You can group multiple element selectors to apply the same styles to multiple elements.

Example :-

```
css
/* Targets all <h1>, <h2>, and <h3> elements */
h1, h2, h3 {
  color: blue;
}
```

In this example, the styles defined inside the curly braces will be applied to all `<h1>`, `<h2>`, and `<h3>` elements. This allows you to apply the same color (blue in this case) to all three heading elements.

2. **Class Group Selectors :-** You can also group multiple class selectors to apply the same styles to elements with different classes.

Example:

```
css
/* Targets all elements with class "highlight" and class "important" */
.highlight, .important {
  background-color: yellow;
}
```

In this example, any element with either the "highlight" class or the "important" class will have a yellow background color applied to it.

3. **ID Group Selectors:** Grouping ID selectors is not common because IDs are unique and should only be used once per page. However, you can still group them to apply shared styles if needed.

Example:

```
css
/* Targets the elements with ID "logo" and ID "banner" */
#logo, #banner {
  width: 200px;
  height: 150px;
}
```

In this example, the styles specified will be applied to both the element with ID "logo" and the element with ID "banner".

Group selectors offer a convenient way to apply shared styles to multiple elements without duplicating the CSS code. They can be used with element selectors, class selectors, ID selectors, or any combination thereof. By grouping selectors,

you can make your CSS code more concise and maintainable.

```
css
/* Targets all <h1> elements, elements with class "highlight", and the elem
h1, .highlight, #logo {
  color: blue;
  font-weight: bold;
}
```

In this example, the styles defined inside the curly braces will be applied to all <h1> elements, elements with the "highlight" class, and the element with the ID "logo". The text color will be blue, and the font weight will be bold for all these elements.

COMMENT IN CSS

In C++, comments are used to add explanatory notes or remarks within the source code. They are ignored by the compiler and do not affect the execution of the program. Comments are helpful for documenting code, providing explanations, and making the code more readable and understandable for developers.

There are two types of comments in C++ :-

1. **Single-line comments:** Single-line comments begin with two forward slashes (//) and continue until the end of the line. They are used to comment on a single line of code or provide a brief explanation.

Example :-

```
cpp
// This is a single-line comment
int x = 5; // Assigning a value to the variable x
```

In the above example, the first line is a single-line comment that provides a general comment. The second line includes a comment at the end of the line to explain the purpose of the code.

2. **Multi-line comments :-** Multi-line comments, also known as block comments, begin with /* and end with */. They can span multiple lines and are useful for adding detailed explanations or temporarily disabling blocks of code.

Example :-

```
cpp
/* This is a multi-line comment
   It can span multiple lines
   Useful for longer explanations or disabling code temporarily */
int y = 10;
```

In this example, the comment block provides a more detailed explanation that can span multiple lines.

Comments are not executed or compiled, so they do not affect the behavior or performance of the program. They are solely meant for human readability and understanding. It is good practice to include comments in your code to enhance its clarity and facilitate future maintenance and collaboration.

JAVASCRIPT

JavaScript is a high-level, interpreted programming language primarily used for developing dynamic and interactive web applications.

JavaScript is a versatile language that can be executed on the client side (in a web browser) as well as on the server side (using frameworks like Node.js). It is known for its ability to manipulate the content, structure, and behavior of web pages in response to user interactions.

Here are some key features and characteristics of JavaScript :-

1. **Client-Side Scripting:** JavaScript is primarily used for client-side scripting, meaning it runs in the user's web browser. It enables web developers to create dynamic and interactive web pages by handling events, and modifying the content and appearance of web elements.
2. **Object-Oriented Programming:** JavaScript is an object-oriented programming (OOP) language, allowing developers to create and manipulate objects that represent real-world entities. It supports concepts

such as encapsulation, inheritance, and polymorphism.

3. **Interactivity and Event Handling:**

JavaScript facilitates user interactions on web pages. It can respond to events like button clicks, form submissions, mouse movements, and keyboard input. Through event handling, developers can trigger specific actions or execute code in response to these events.

4. **Cross-Browser Compatibility:**

JavaScript is supported by all major web browsers, including Chrome, Firefox, Safari, and Edge. This allows developers to write code that works consistently across different browsers and platforms.

5. **Integration with HTML and CSS:**

JavaScript can be seamlessly integrated with HTML and CSS, the other core technologies of web development. It enables dynamic modification of HTML elements, manipulation of CSS styles, and the creation of interactive user interfaces.

6. **Extensibility and Libraries:**

JavaScript has a vast ecosystem of libraries and frameworks that extend its capabilities and simplify common tasks. Popular libraries include jQuery, React.js, AngularJS, and Vue.js, among others.

JavaScript's versatility and widespread adoption make it a crucial language for web development. It enables developers to create interactive user experiences, validate form inputs, perform data manipulation, communicate with servers, and build complex web applications.

WHAT CAN JAVASCRIPT DO ?

Things that should be done every time a page loads: When a web page loads, JavaScript can be used to perform various tasks automatically. This can include initializing variables, setting up event listeners, fetching data from a server, manipulating the page structure or content, and much more. For example, you can use the

`window.onload` event handler to execute code once the entire page has finished loading :-

This is useful for tasks like initializing the state of a web application, making AJAX requests for data, or attaching event handlers to specific elements on the page.

Things that should be done when the page is closed: JavaScript also provides the ability to execute code when a web page is about to be closed or navigated away from. This can be useful for performing cleanup tasks or saving user data before leaving the page. **This event handler allows you to prompt the user with a confirmation dialog before they leave the page :-**

Here, you can include any necessary code to handle data saving, clean up resources, or display a confirmation message to the user.

Action that should be performed when a user clicks a button: JavaScript allows you to define event handlers that respond to user actions, such as clicking a button. By attaching an event handler function to the button's onclick event, you can specify the code that should run when the button is clicked. For example :-

Here, you can define the desired action or behavior that should occur when the button is clicked, such as displaying a message, validating input, making AJAX requests, or updating the page content.

Content that should be verified when a user inputs data: JavaScript can also be used to verify and validate user input in real-time. By attaching event handlers to input fields, you can respond to user interactions, validate the entered data, and provide feedback or error messages. For example, you can use the on input event to check the validity of an email input field as the user types :-

Here, you can write code to validate the entered email value against a specific pattern or perform other checks. You can then update the UI to indicate whether the input is valid or display error messages if necessary.

By utilizing event handlers in JavaScript, you can control and respond to various user interactions and browser actions, ensuring a dynamic and interactive user experience on your web pages or application

JavaScript Program Structure: A JavaScript program consists of variables, objects, and functions. It is a high-level programming language that follows an object-oriented programming (OOP) approach.

Terminating Statements and Blocks: In JavaScript, each statement is terminated by a semicolon (;). Blocks of code are enclosed within curly brackets {} to define their scope and separate them from other code blocks.

Functions and Parameters: Functions in JavaScript are defined using the function keyword, followed by the function name and a pair of parentheses (). Parameters (if any) are declared inside the parentheses. Functions are a way to encapsulate reusable code blocks.

Variables: Variables in JavaScript are declared using the var keyword. They can hold various types of values, including numbers, strings, booleans, objects, and more. Variables are used to store and manipulate data within a program.

Script Placement: JavaScript code can be implemented within the <script>...</script> HTML tags in a web page. The <script> tags are used to alert the browser to interpret the enclosed text as JavaScript code.


Placement Recommendation: It is typically recommended to place the <script> tags containing JavaScript code within the <head> tags of the HTML document. This allows the JavaScript code to be loaded and executed before the body content is rendered, ensuring proper functionality.

POSITION OF JAVASCRIPT

JavaScript is a programming language used for adding interactivity and dynamic functionality to web pages. It can be embedded directly in an HTML document using the <script> tag. Here's

a breakdown of the different ways you can include JavaScript code in your HTML:


Inline JavaScript: You can include JavaScript code directly within the <script> tags in your HTML document. This can be done within the <head> or <body> section of the HTML.



```
html
Copy code
<script>
// JavaScript code goes here
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

External JavaScript: JavaScript code can be placed in external files with a .js extension. This allows you to separate your JavaScript code from the HTML, making it easier to manage and reuse.

Example: myScript.js



```
javascript
Copy code
function myFunction() {
document.getElementById("demo").innerHTML = "Ethio Programming.";
}
```

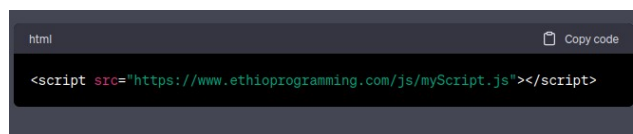
To use an external JavaScript file, include the <script> tag with the src attribute pointing to the file.

<script src="myScript.js"> </script>

You can place the external script reference in the <head> or <body> section as per your preference.

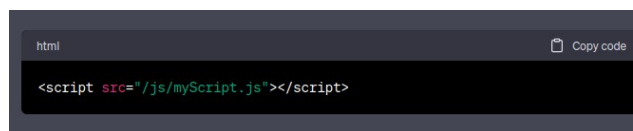
External References: External JavaScript files can be referenced using a full URL or a relative path to the current web page.

Example using full URL:



```
html
Copy code
<script src="https://www.ethioprogramming.com/js/myScript.js"></script>
```

Example using a relative path:



```
html
Copy code
<script src="/js/myScript.js"></script>
```

Including JavaScript in your HTML allows you to add interactivity, manipulate HTML elements, handle events, and perform various operations on the client-side. Using external JavaScript files promotes code reusability, separates concerns, and makes the HTML and JavaScript code easier to read and maintain.

JAVASCRIPT COMMENTS

In JavaScript, comments are used to add explanatory notes or to disable certain lines of code. Comments are ignored by the JavaScript interpreter and do not affect the execution of the code. They are purely for human readability and understanding.

JavaScript supports two types of comments:

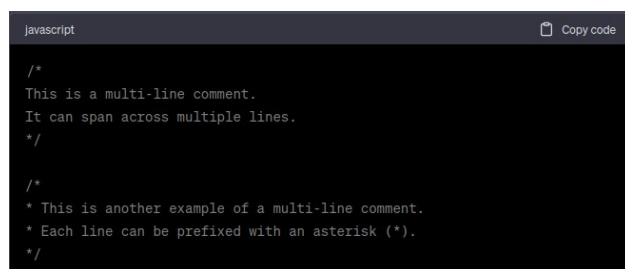
1. **Single-line comments:** Single-line comments start with two forward slashes (`//`). Everything after the `//` on the same line is treated as a comment and is ignored by the interpreter.

Example:

```
// This is a single-line comment
```

Multi-line comments: Multi-line comments are enclosed between `/*` and `*/` delimiters. They can span across multiple lines and are useful for longer explanations or for temporarily disabling blocks of code.

Example:

A screenshot of a code editor with a dark theme. The editor shows two examples of multi-line comments. The first example is a block of text: `/* This is a multi-line comment. It can span across multiple lines. */`. The second example is a block of text: `/* * This is another example of a multi-line comment. * Each line can be prefixed with an asterisk (*). */`. The editor has a tab labeled 'javascript' and a 'Copy code' button in the top right corner.

Comments are commonly used to:

- Explain the purpose or functionality of a particular code segment.
- Provide documentation and improve code readability for other developers.
- Disable specific lines or blocks of code temporarily during testing or debugging.

- Add reminders or TODO notes for future enhancements or fixes.

It is good practice to use comments effectively to make your JavaScript code more understandable and maintainable.

In JavaScript, variables are used to store and manipulate data. They serve as containers that hold values, which can be of various types such as numbers, strings, booleans, objects, or functions. Variables in JavaScript are dynamically typed, meaning they can hold values of any type, and the type of a variable can be changed at any time.

To declare a variable in JavaScript, you use the `var`, `let`, or `const` keyword, followed by the variable name. Here's the syntax:

In JavaScript, variables are used to store and manipulate data. They serve as containers that hold values, which can be of various types such as numbers, strings, booleans, objects, or functions. Variables in JavaScript are dynamically typed, meaning they can hold values of any type, and the type of a variable can be changed at any time.

To declare a variable in JavaScript, you use the `var`, `let`, or `const` keyword, followed by the variable name. Here's the syntax:

A screenshot of a code editor with a dark theme. The editor shows three examples of variable declarations: `var variableName;`, `let variableName;`, and `const variableName;`. The editor has a tab labeled 'javascript' and a 'Copy code' button in the top right corner.

`var` is the oldest way to declare a variable. It has function scope or global scope, meaning it is accessible within the function or throughout the entire program.

- `let` was introduced in ES6 (ECMAScript 2015) and provides block scope. It is limited to the block it is declared in, such as within a function or loop.
- `const` also came with ES6 and is used to declare constants. Once assigned a value, the value cannot be changed.

Examples of variable declarations and assignments:

```
javascript Copy code
var age; // Declaring a variable without assigning a value
age = 25; // Assigning a value to the variable

let name = "John"; // Declaring and assigning a value in one line

const PI = 3.14; // Declaring and assigning a constant value

var x = 10, y = 20, z = 30; // Declaring multiple variables in one statement
```

Variables can be used in expressions, assigned new values, and their values can be retrieved or modified. Here are some common operations:

```
javascript Copy code
var x = 5;
var y = 10;
var sum = x + y; // Performing addition

var message = "Hello, ";
var name = "John";
var greeting = message + name; // Concatenating strings

x = 7; // Assigning a new value to an existing variable

console.log(x); // Outputting the value of a variable to the console
```

JavaScript also provides shorthand notations for updating variable values:

```
javascript Copy code
var counter = 0;
counter += 1; // Incrementing the variable value by 1
counter -= 1; // Decrementing the variable value by 1
counter *= 2; // Multiplying the variable value by 2
counter /= 2; // Dividing the variable value by 2
```

It's important to note that JavaScript is case-sensitive, so `myVariable` and `myvariable` are treated as separate variables. It's a good practice to use meaningful and descriptive names for variables to enhance code readability.

Overall, variables in JavaScript play a crucial role in storing and manipulating data, allowing you to create dynamic and interactive web applications.

FORM VALIDATION IN JAVASCRIPT

Data validation is the process of checking and ensuring that user input is accurate, complete, and suitable for its intended use. It helps ensure that the data entered by users is valid and meets the expected criteria. Common validation tasks include checking if all required fields are filled,

verifying if a date is entered correctly, or ensuring that numeric fields only contain numeric values.

Validation can be performed using different methods and can be implemented on both the server side and the client side.

Server-side validation is done on the web server after the user submits the data. It involves validating the input data on the server to ensure it meets the specified requirements before processing or storing it. This type of validation provides a more secure and reliable approach since the server has control over the data processing.

Client-side validation is performed on the user's web browser before the data is sent to the server. It offers immediate feedback to the user and can improve the user experience by preventing unnecessary server requests. However, client-side validation should not be solely relied upon, as it can be bypassed or manipulated by the user. Server-side validation should always be used in conjunction with client-side validation to ensure data integrity.

Example :- In this example, we have a form with three input fields: "Name," "Email," and "Password." The JavaScript function `validateForm()` is called when the form is submitted.

Inside the `validateForm()` function, we retrieve the values of the name, email, and password fields using `document.forms["myForm"]["name"].value`, `document.forms["myForm"]["email"].value`, and `document.forms["myForm"]["password"].value`, respectively.

We then perform the following validations:

- Check if the name field is empty. If so, display an alert message saying "Name is required" and return false to prevent form submission.
- Check if the email field is empty and validate its format using a regular expression. If the email is empty, display an alert message saying "Email is

required." If the email format is invalid, display an alert message saying "Invalid email format." Return false in both cases to prevent form submission.

- Check if the password field is at least 8 characters long. If the password is less than 8 characters, display an alert message saying "Password must be at least 8 characters long" and return false to prevent form submission.

If all the validations pass, we display an alert message saying "Form submitted successfully!" and return true to allow the form to be submitted.

This example covers basic form validation tasks such as checking for required fields, validating email format, and ensuring a minimum password length.

PHP

Variables in PHP

In simpler terms, variables in PHP are like containers that can hold different types of data. You can give a variable a name and assign a value to it using the assignment operator (=).

In PHP, you create a variable by starting with the dollar sign (\$) followed by the variable name. For example, \$txt is a variable that can hold a string, \$x can hold an integer, and \$y can hold a floating-point number.

Unlike some other programming languages, you don't need to explicitly declare a variable in PHP. It is created automatically when you assign a value to it.

There are some rules for naming variables in PHP:

- A variable name must start with a letter or an underscore.
- A variable name cannot start with a number.
- A variable name can only contain letters, numbers, and underscores.
- Variable names are case-sensitive, so \$age and \$AGE are considered as two different variables.

You can output the value of a variable using the echo statement. It allows you to display text along with the value of a variable. For example, echo "I am following \$txt!"; will output "I am following Ethio Programming!".

One important thing to note is that PHP is a loosely typed language, which means you don't

```
html
Copy code

<!DOCTYPE html>
<html>
<head>
  <title>Form Validation Example</title>
  <script>
    function validateForm() {
      // Get form inputs
      var name = document.forms["myForm"]["name"].value;
      var email = document.forms["myForm"]["email"].value;
      var password = document.forms["myForm"]["password"].value;

      // Perform validation

      // Check if name is empty
      if (name === "") {
        alert("Name is required");
        return false;
      }

      // Check if email is empty and valid format
      if (email === "") {
        alert("Email is required");
        return false;
      } else if (!validateEmail(email)) {
        alert("Invalid email format");
        return false;
      }

      // Check if password is at least 8 characters long
      if (password.length < 8) {
        alert("Password must be at least 8 characters long");
        return false;
      }

      // All validations passed, form can be submitted
      alert("Form submitted successfully!");
      return true;
    }

    // Function to validate email format using regular expression
    function validateEmail(email) {
      var re = /\S+@\S+\.\S+/;
      return re.test(email);
    }
  </script>
</head>
<body>
  <h1>Form Validation Example</h1>
  <form name="myForm" onsubmit="return validateForm()">
    <label for="name">Name:</label>
    <input type="text" name="name" required><br>

    <label for="email">Email:</label>
    <input type="email" name="email" required><br>

    <label for="password">Password:</label>
    <input type="password" name="password" required><br>

    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Regenerate response

need to explicitly specify the data type of a variable. PHP automatically determines the data type based on the value assigned to the variable. This is different from languages like C, C++, and Java, where you have to declare the variable's name and type before using it.

```
php
Copy code
<?php
$txt = "I love Ethiopia!!!"; // Assigning a string value to the variable $txt
$num1 = 78; // Assigning an integer value to the variable $num1
$num2 = 67.43; // Assigning a floating-point value to the variable $num2

echo $txt . " " . $num1; // Outputting the value of $txt, followed by a space and the value of $num1
?>
```

In the first line, we declare a variable named `$txt` and assign it the value "I love Ethiopia!!!". This variable holds a string.

- In the second line, we declare a variable named `$num1` and assign it the value 78. This variable holds an integer.
- In the third line, we declare a variable named `$num2` and assign it the value 67.43. This variable holds a floating-point number.
- The echo statement is used to output the value of variables. In this case, we concatenate (join together) the value of `$txt`, a space " ", and the value of `$num1`. The dot (.) is used for concatenation.
- When the code is executed, it will display the following output: "I love Ethiopia!!! 78". The value of `$txt` is displayed first, followed by a space, and then the value of `$num1`.

So, the code assigns values to variables and then outputs a string and an integer value using the echo statement.

```
php
Copy code
<?php
$txt = "I love Ethiopia!!!"; // Assigning a string value to the variable $txt
$num1 = 78; // Assigning an integer value to the variable $num1
$num2 = 67.43; // Assigning a floating-point value to the variable $num2

echo "Hello all $txt"; // Outputting the string "Hello all" followed by the value of $txt
?>
```

- In the first line, we declare a variable named `$txt` and assign it the value "I love Ethiopia!!!". This variable holds a string.

- In the second line, we declare a variable named `$num1` and assign it the value 78. This variable holds an integer.
- In the third line, we declare a variable named `$num2` and assign it the value 67.43. This variable holds a floating-point number.
- The echo statement is used to output a message. In this case, we output the string "Hello all" followed by the value of `$txt`. The variable `$txt` is embedded within the string using double quotes and the variable name is preceded by a dollar sign (\$).
- When the code is executed, it will display the following output: "Hello all I love Ethiopia!!!". The string "Hello all" is displayed first, followed by the value of `$txt`.

PHP VARIABLE SCOPE

In PHP, there are three different variable scopes: **local**, **global**, and **static**. Here's an explanation of each:

1. Local Scope:

- Variables declared within a function are considered to have a local scope.
- Local variables can only be accessed and used within the function in which they are declared.
- They are not accessible outside the function or in other functions unless explicitly passed as arguments or returned from the function.

2. Global Scope:

- Variables declared outside of any function, at the top-level of the script, have a global scope.
- Global variables can be accessed and used from anywhere within the script, including inside functions.
- To use a global variable inside a function, you need to explicitly declare it as global within the function before using it.

3. Static Scope:

- Static variables are a special type of local variables that retain their values between function calls.
- Unlike regular local variables, static variables do not lose their values when the function execution ends.
- The variable is initialized only once, on the first execution of the function, and retains its value in subsequent calls.
- Static variables are often used when you need to maintain some information or state across multiple function calls.

Imagine you have a special type of box called a "static box." This box is different from regular boxes because it can remember its contents even after you close it. Every time you open the box, you will find the same things inside that you put in the first time.

In programming, static variables work similarly. They are like special boxes that remember their values between different function calls. When a function finishes executing, regular variables lose their values and are reset. But static variables are like those special boxes that keep their contents intact even after the function ends.

When you use a static variable in a function, it is initialized only once, the first time the function is called. After that, it retains its value in every subsequent call. This is useful when you want to maintain some information or keep track of a state that needs to be remembered across multiple function calls.

So, static variables provide a way to store and remember values that can be accessed and modified each time a function is called, ensuring that the value is retained for future use.

```
cpp
#include <iostream>

void printCount() {
    static int count = 0; // static variable

    // Increment the count and print its value
    count++;
    std::cout << "Count: " << count << std::endl;
}

int main() {
    // Call the printCount function multiple times
    printCount(); // Output: Count: 1
    printCount(); // Output: Count: 2
    printCount(); // Output: Count: 3

    return 0;
}
```

In this example, we have a function called printCount() that contains a static variable named count. The printCount() function increments the value of count by 1 and prints its current value.

When we call the printCount() function multiple times from the main() function, the static variable count retains its value between the function calls. It is initialized only once, on the first execution of the printCount() function. In subsequent calls, the value of count is not reset to 0 but rather continues from where it left off.

So, when we run the program, we will see the count increasing by 1 each time the printCount() function is called. The static variable count remembers its value across different calls to the function.

```
Output:
makefile
Count: 1
Count: 2
Count: 3
```

Now, let's look at the code you provided and explain the issue related to variable scoping:

```
php
<?php
$x = 0;
$y = 0;
function add()
{
    echo $x + $y;
}
echo $y;
add();
?>
```

In this code, you have declared two global variables `$x` and `$y` outside the function. Then, inside the `add()` function, you are trying to access and use these variables. However, you will encounter an error because within the function's scope, the variables `$x` and `$y` are not recognized as global variables automatically.

To fix this, you need to explicitly declare them as global within the function using the `global` keyword:

```
php
function add()
{
    global $x, $y;
    echo $x + $y;
}
```

ECHO AND PRINT STATEMENT

Let's define the differences between the `echo` and `print` statements in more detail:

Return Value:

`echo` has no return value. It simply outputs the specified content to the screen. `print` has a return value of `1`, which allows it to be used in expressions. This means that you can assign the value returned by `print` to a variable or use it within an expression.

Parameter Handling:

`echo` can accept multiple parameters, separated by commas. This means that you can pass multiple values or variables to `echo` to be displayed. `print` can only accept a single argument. It does not support multiple parameters. If you try to pass multiple values to `print`, you will encounter a syntax error.

Speed:

In terms of performance, `echo` is marginally faster than `print`. The difference in speed is usually negligible, but in large-scale applications with heavy usage of output statements, this slight performance advantage of `echo` can make a difference.

Overall, both `echo` and `print` serve the same purpose of outputting data to the screen. They are interchangeable in most cases, and the choice between them often comes down to personal preference or specific use cases. If you need to output multiple values or variables, `echo` is more suitable. If you require the return value for expressions, `print` can be used.

```
php
Copy code

<?php
// Example 1: Using echo statement
echo "<h2>PHP is Fun!</h2>"; // Output: PHP is Fun!
echo "Hello world!<br>"; // Output: Hello world!
echo "I'm about to learn PHP!<br>"; // Output: I'm about to learn PHP!
echo "This ", "string ", "was ", "made ", "with multiple parameters."; // Ou
?>

<?php
// Example 2: Using print statement
print "<h2>In this tutorial, we will learn about PHP Programming</h2>"; // O
print "Hello Guys, This is Ethio Programming!<br>"; // Output: Hello Guys, T
print "Stay tuned with us!"; // Output: Stay tuned with us!
?>

<?php
// Example 3: Using echo and print statements
echo ("Hello Ethio Programming followers!!<br>"); // Output: Hello Ethio Pro
echo "Hello Ethio Programming followers!!<br>"; // Output: Hello Ethio Progi
print ("Hello Ethio Programming followers!!<br>"); // Output: Hello Ethio Pr
print "Hello Ethio Programming followers!!"; // Output: Hello Ethio Programm
?>
```

In the above examples, we are using the `echo` and `print` statements to output text to the screen. Both statements can be used to display text or HTML markup. The main difference between `echo` and `print` is that `echo` does not have a return value, while `print` has a return value of `1`, which allows it to be used in expressions.

In Example 1, we use the `echo` statement to output various lines of text and HTML markup. Each statement is enclosed in double quotes (`"`) and can be concatenated using the dot (`.`) operator.

In Example 2, we use the `print` statement to achieve the same output as Example 1. Similar to `echo`, we can include HTML markup or plain text within the `print` statement. However, note that `print` can only accept a single argument.

In Example 3, we demonstrate the usage of both `echo` and `print` statements interchangeably. We show that parentheses are optional for both `echo` and `print`.

When running the code, the text and HTML markup specified in the echo or print statements will be displayed on the screen, producing the expected output.

DATA TYPE IN PHP

In PHP, data types specify the type of data that a variable can hold. PHP supports several data types, including strings, integers, floats, booleans, arrays, objects, NULL, and resources. Understanding these data types is essential for working with data and performing various operations in PHP.

PHP Data Types:

String: A string is a sequence of characters enclosed in quotes. It can be created using single quotes (') or double quotes ("). Strings can contain letters, numbers, symbols, and special characters. Here's an example:

```
php
Copy code
$x = "Hello world!";
$y = 'Hello world!';
echo $x;
echo "<br>";
echo $y;
```

Integer: An integer is a non-decimal number without a decimal point. It can be either positive or negative. PHP integers can be specified in decimal, hexadecimal, or octal formats. Here's an example:

```
$x = 5985;
var_dump($x);
```

Float: A float, also known as a floating-point number, is a number with a decimal point or in exponential form. It represents fractional values. Here's an example:

```
$x = 10.365;
var_dump($x);
```

Boolean: A boolean represents two possible states: TRUE or FALSE. It is often used for conditional statements and logical operations. Here's an example:

```
$x = true;
```

```
$y = false;
```

Array: An array is a collection of values stored in a single variable. It allows you to store multiple values of different data types in an ordered or associative manner. Here's an example:

```
$cars = array("Volvo", "BMW", "Toyota");
var_dump($cars);
```

NULL: NULL is a special data type that represents the absence of a value. It is often used to indicate that a variable has no value assigned to it. Here's an example:

```
$x = null;
var_dump($x);
```

In the examples above, the var_dump() function is used to display the data type and value of the variables. It helps in understanding the data type of a variable and its current value.

By utilizing these different data types, you can store, manipulate, and process various kinds of data in PHP, making your code more flexible and powerful.

STRING IN PHP

GET THE LENGTH OF A STRING

In PHP, a string is a sequence of characters, such as "Hello world!". Strings are commonly used to store and manipulate text data in PHP.

To determine the length of a string, PHP provides the strlen() function. The strlen() function counts the number of characters in a given string and returns that value.

For example, if we have the string "Hello world!", we can use the strlen() function to find its length and display it:

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

In this example, the strlen("Hello world!") statement returns the value 12 because the string "Hello world!" consists of 12 characters. The echo statement then displays the result.

By using the `strlen()` function, you can easily determine the length of a string in PHP, which can be helpful for various string manipulation tasks.

COUNT THE NUMBER OF STRINGS IN A STRING

In PHP, you can count the number of words in a string using the `str_word_count()` function. This function takes a string as input and returns the count of words present in that string.

For example, let's consider the string "Hello world!". We can use the `str_word_count()` function to count the number of words in it:

```
<?php
echo str_word_count("Hello world!"); // outputs
2 ?>
```

In this example, the `str_word_count("Hello world!")` statement returns the value 2 because the string "Hello world!" contains two words, "Hello" and "world". The echo statement then displays the result.

The `str_word_count()` function is useful when you need to count the number of words in a string, which can be helpful in tasks such as text analysis, word processing, or word frequency calculations.

REVERSE A STRING

In PHP, you can reverse a string using the `strrev()` function. This function takes a string as input and returns the reversed version of that string.

For example, let's consider the string "Hello world!". We can use the `strrev()` function to reverse it:

```
<?php
echo strrev("Hello world!"); // outputs !dlrow
olleH ?>
```

In this example, the `strrev("Hello world!")` statement returns the reversed version of the string "Hello world!", which is "!dlrow olleH". The echo statement then displays the reversed string.

The `strrev()` function is useful when you need to reverse a string, such as when you want to display a string in reverse order or perform specific operations that require the reversed string.

SEARCH FOR A SPECIFIC TEXT WITH IN A STRING

In PHP, you can search for a specific text within a string using the `strpos()` function. This function takes two parameters: the string to search within and the text you want to find.

For example, let's consider the string "Hello world!". We can use the `strpos()` function to search for the text "world" within this string:

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

In this example, the `strpos("Hello world!", "world")` statement searches for the text "world" within the string "Hello world!". Since "world" is found starting at the 6th character of the string, the function returns the character position 6. The echo statement then displays this result.

If the text you are searching for is not found within the string, the `strpos()` function will return FALSE. This can be useful for checking if a specific text exists within a string.

The `strpos()` function is commonly used when you need to locate the position of a specific substring within a larger string and perform certain operations based on its position.

REPLACE TEXT WITH IN A STRING

In PHP, you can replace specific characters or text within a string using the `str_replace()` function. This function takes three parameters: the characters or text you want to replace, the replacement characters or text, and the original string.

For example, let's consider the string "Hello world!". We can use the `str_replace()` function to

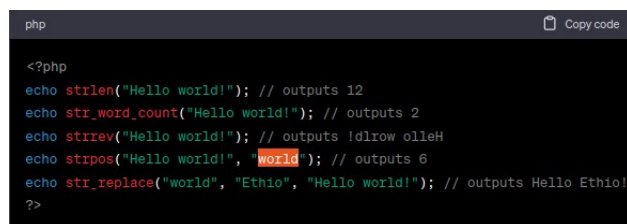
replace the text "world" with "Ethio" within this string:

```
<?php
echo str_replace("world", "Ethio", "Hello
world!"); // outputs Hello Ethio!
?>
```

In this example, the `str_replace("world", "Ethio", "Hello world!")` statement replaces the text "world" with "Ethio" within the string "Hello world!". The resulting string becomes "Hello Ethio!". The `echo` statement then displays this modified string.

The `str_replace()` function is useful when you need to replace specific characters or text patterns within a string with something else. It can be handy for tasks like find-and-replace operations or modifying the content of a string based on certain criteria.

Example :-



```
php
Copy code

<?php
echo strlen("Hello world!"); // outputs 12
echo str_word_count("Hello world!"); // outputs 2
echo strrev("Hello world!"); // outputs !dlrow olleH
echo strpos("Hello world!", "world"); // outputs 6
echo str_replace("world", "Ethio", "Hello world!"); // outputs Hello Ethio!
?>
```

This block of PHP code is embedded within the HTML document using the `<?php ?>` tags. Inside the PHP code, several functions are used:

- `strlen("Hello world!")` returns the length of the string "Hello world!", which is 12.
- `str_word_count("Hello world!")` counts the number of words in the string "Hello world!", which is 2.
- `strrev("Hello world!")` reverses the string "Hello world!", resulting in "!dlrow olleH".
- `strpos("Hello world!", "world")` searches for the text "world" within the string "Hello world!" and returns the position of the first occurrence, which is 6.

- `str_replace("world", "Ethio", "Hello world!")` replaces the text "world" with "Ethio" in the string "Hello world!", resulting in "Hello Ethio!".

REQUIRE AND INCLUDE IN PHP

The PHP include statement is used to include the content of another file within the current PHP file. This allows you to reuse common code or markup across multiple pages of a website. The included file can contain PHP code, HTML, or plain text.

There are two similar statements in PHP for including files: **include** and **require**. The main difference between them is how they handle failures:

1. **include:** If the included file is not found, PHP will generate a warning (E_WARNING) but continue executing the script.
2. **require:** If the included file is not found, PHP will generate a fatal error (E_COMPILE_ERROR) and stop executing the script.

It is recommended to use `require` when including key files that are necessary for the flow of execution, as it ensures that the script doesn't continue if the required file is missing. For less critical files, such as reusable components like headers, footers, or menus, you can use `include`.

The syntax for including files is as follows:

include 'filename';

require 'filename';

In the above syntax, 'filename' should be replaced with the path and name of the file you want to include. This can be a relative or absolute path depending on the location of the file.

Including files can greatly simplify your code and make it more maintainable. For example, you can create a header.php file that contains the header markup and PHP code, and then include it in all your web pages. This way, if you need to update

the header, you only need to modify the header.php file, and the changes will be reflected across all pages that include it.

Here's an example of including a file named header.php:

```
php Copy code
<!DOCTYPE html>
<html>
<head>
  <title>My Website</title>
  <link rel="stylesheet" href="styles.css">
  <?php include 'header.php'; ?>
</head>
<body>
  <h1>Welcome to my website</h1>
  <?php include 'navigation.php'; ?>
  <p>This is the content of my website.</p>
  <?php include 'footer.php'; ?>
</body>
</html>
```

In the example above, the header.php, navigation.php, and footer.php files are included within the HTML structure. This allows you to separate the common elements of your website into separate files for easier maintenance and reusability.

Remember to use the appropriate include statement (include or require) based on the importance of the included file to your application's execution flow.

USING VARIABLES FROM THE INCLUDED FILE

When you include a file in PHP, any variables defined within that file become accessible in the file that includes it. This allows you to use and manipulate those variables as if they were defined directly in the including file.

Here's how it works in detail:

1. **Including the File:** To include a file and make its variables accessible, you can use the include or require statement followed by the file path. For example:

```
include 'variables.php';
```

This includes the file named variables.php and makes its variables available in the current file.

2. **Variable Scope:** PHP follows a concept called variable scope, which determines

where a variable is accessible and can be used. When you include a file, the included file inherits the variable scope of the file that includes it.

3. **Accessing Variables:** Once the file is included, you can access the variables defined within it using their names. For example, if the included file defines a variable named \$name, you can use it in the including file as if it were defined there directly. Here's an example:

```
include 'variables.php';
```

```
echo "Hello, " . $name . "!"; // Accessing the $name variable from the included file
```

In this example, the \$name variable from the variables.php file is accessed and concatenated with the string "Hello, " to produce the output.

4. **Modifying Variables:** You can also modify the included variables within the including file. Any changes made to the variables will affect their values in the included file as well. For example:

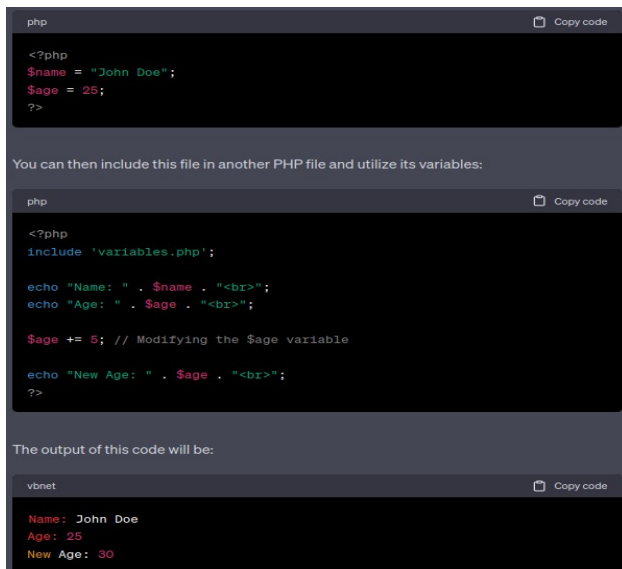
```
include 'variables.php';
```

```
$age += 5; // Modifying the $age variable from the included file
```

In this example, the \$age variable from the variables.php file is incremented by 5 within the including file.

It's important to note that including files should be done with caution to avoid naming conflicts or unexpected behavior. Ensure that the included file does not redefine variables that are already in use in the including file to avoid conflicts.

Example: Assume you have an variables.php file with the following contents :-



```
php
Copy code

<?php
$name = "John Doe";
$age = 25;
?>

You can then include this file in another PHP file and utilize its variables:

php
Copy code

<?php
include 'variables.php';

echo "Name: " . $name . "<br>";
echo "Age: " . $age . "<br>";

$age += 5; // Modifying the $age variable

echo "New Age: " . $age . "<br>";
?>

The output of this code will be:

vbnet
Copy code

Name: John Doe
Age: 25
New Age: 30
```

PHP DATABASE CONNECTIVITY

MySQL is a popular database management system used in web development to store and manage data. It is a server-based database system, meaning it runs on a server and can be accessed by multiple clients.

MySQL organizes data in tables, which are structured collections of related data. Each table consists of columns (fields) and rows (records). For example, in a company's database, there may be tables for employees, products, customers, and orders. The tables allow for efficient storage, retrieval, and manipulation of data in a structured manner.

Using MySQL, developers can create databases, define tables with appropriate columns and data types, perform queries to retrieve specific data, insert new records, update existing records, and delete unwanted data. It provides a robust and reliable foundation for managing data in web applications.

CREATING A CONNECTION

When developing web applications, it is common to interact with a MySQL database to store and retrieve data. PHP provides built-in functions and extensions to establish a connection between PHP and MySQL, allowing you to perform database operations seamlessly.

Establishing a Connection with PHP and MySQL:

To establish a connection between PHP and MySQL, you need the following information:

1. **Hostname:** The server name where the MySQL database is hosted.
2. **Username:** The username used to access the MySQL database.
3. **Password:** The password associated with the username.
4. **Database Name:** The name of the specific database you want to connect to.

Once you have this information, you can use the following steps to establish a connection :

1. **Open a PHP script:** Start by opening a PHP script where you will write your code.
2. **Establish the connection:** Use the `mysqli_connect()` function to establish a connection between PHP and MySQL. Pass in the hostname, username, password, and database name as parameters. This function returns a connection object that you can use to interact with the database.
3. **Check the connection:** After establishing the connection, it is essential to verify if the connection was successful. You can use the `mysqli_connect_error()` function to check for any connection errors. If there are no errors, you can proceed with executing your database queries.
4. **Perform database operations:** Once the connection is established and validated, you can execute SQL queries using the connection object. You can use functions like `mysqli_query()` to execute queries and retrieve results from the database.
5. **Close the connection:** After completing your database operations, it is good practice to close the connection using the `mysqli_close()` function. This helps free up system resources and ensures proper termination of the connection.

```

php
Copy code

<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

echo "Connected successfully";
?>

```

In this example, the code establishes a connection to the MySQL database using the provided server name, username, password, and database name. If the connection is successful, it displays the message "Connected successfully." If there is an error connecting to the database, it will display the specific error message using the `mysqli_connect_error()` function.

Remember to replace "your_username", "your_password", and "your_database" with your actual MySQL credentials and database name for the code to work correctly.

CREATE A DATABASE

In PHP, you can use the MySQLi extension to create a new database on a MySQL server. The MySQLi extension provides a set of functions specifically designed for interacting with MySQL databases. To create a database, you need to establish a connection to the MySQL server and execute an SQL query to create the database.

Here's an example code snippet that demonstrates how to create a MySQL database using PHP :-

```

php
Copy code

<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Create a new database
$dbname = "your_new_database";
$sql = "CREATE DATABASE $dbname";
if (mysqli_query($conn, $sql)) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . mysqli_error($conn);
}

// Close the connection
mysqli_close($conn);
?>

```

In the above code, replace "your_username" and "your_password" with your actual MySQL credentials. The code establishes a connection to the MySQL server using the `mysqli_connect` function. If the connection is successful, it executes an SQL query to create a new database with the name specified in the `$dbname` variable.

If the database creation is successful, it displays the message "Database created successfully." Otherwise, it outputs an error message with the specific error details.

Remember to close the connection using `mysqli_close($conn)` to free up resources after you're done with the database operations.

CREATE A TABLE

When working with a MySQL database, you often need to create tables to store your data. Tables consist of columns (fields) and rows (records). Each column represents a specific attribute, and each row represents a set of values for those attributes.

To create a table in MySQL, you can use the `CREATE TABLE` statement. This statement allows you to define the table's structure, including the column names, data types, constraints, and more.

Here's an example code snippet that demonstrates how to create a table named "employee" with

four columns: "id", "firstname", "lastname", "email", and "reg_date". The "id" column is set as the primary key, and the "reg_date" column has the data type of TIMESTAMP:

```
php
Copy code

<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// SQL query to create table
$sql = "CREATE TABLE employee (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    firstname VARCHAR(30) NOT NULL,
    lastname VARCHAR(30) NOT NULL,
    email VARCHAR(50),
    reg_date TIMESTAMP
)";

// Execute the query
if (mysqli_query($conn, $sql)) {
    echo "Table 'employee' created successfully";
} else {
    echo "Error creating table: " . mysqli_error($conn);
}

// Close the connection
mysqli_close($conn);
?>
```

In the above code, replace "your_username", "your_password", and "your_database" with your actual MySQL credentials and the name of your database.

The code establishes a connection to the MySQL server using `mysqli_connect`, and if the connection is successful, it executes the SQL query to create the "employee" table. The result is then checked, and appropriate success or error messages are displayed.

Finally, remember to close the connection using `mysqli_close` to free up resources after you're done with the database operations.

INSERT IN TO A TABLE

Once you have created a database and a table with the desired structure, you can start adding data to the table. The `INSERT INTO` statement is used to insert new records into a MySQL table.

Here are some important points to keep in mind when inserting data:

- The SQL query must be quoted in PHP.
- String values inside the SQL query must be quoted using single quotes (') or double quotes (").
- Numeric values should not be quoted.
- The keyword `NULL` (for inserting `NULL` values) should not be quoted.

Now let's see an example code snippet that demonstrates how to insert a new record into the "employee" table:

```
php
Copy code

<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// SQL query to insert data
$sql = "INSERT INTO employee (firstname, lastname, email)
VALUES ('Tola', 'Adamu', 'tolaasamu@gmail.com')";

// Execute the query
if (mysqli_query($conn, $sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

// Close the connection
mysqli_close($conn);
?>
```

In the above code, replace "your_username", "your_password", and "your_database" with your actual MySQL credentials and the name of your database.

The code establishes a connection to the MySQL server using `mysqli_connect`, and if the connection is successful, it executes the SQL query to insert a new record into the "employee" table. The result is then checked, and appropriate success or error messages are displayed.

Remember to adjust the column names and the corresponding values in the `VALUES` clause according to your table structure and the data you want to insert.

Finally, don't forget to close the connection using `mysqli_close` to free up resources after you're done with the database operations.

Note that if a column is set as `AUTO_INCREMENT` (like the "id" column in our example) or `TIMESTAMP`, you don't need to include it in the column list or specify a value in the SQL query. MySQL will automatically handle it for you.

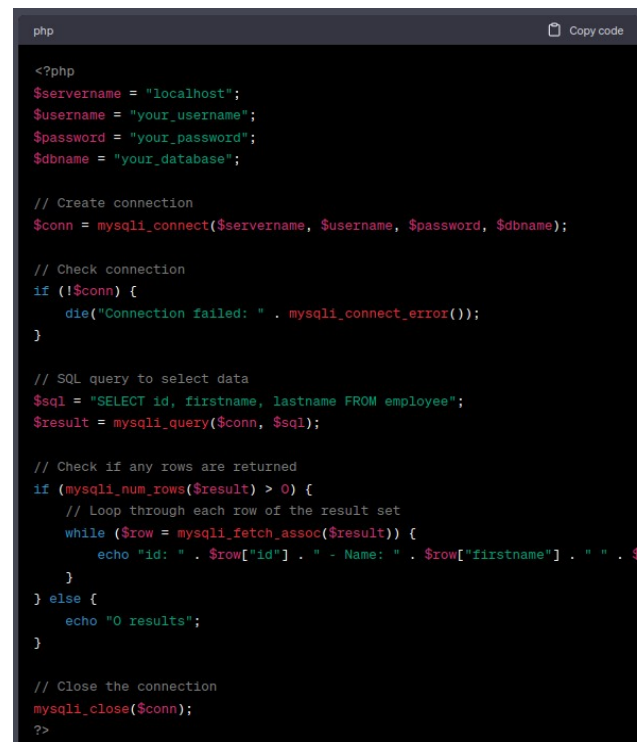
SELECT STATEMENT

Once you have established a connection to the MySQL database, you can use the `SELECT` statement to retrieve data from one or more tables. The `SELECT` statement allows you to specify the columns you want to fetch from the table.

Here are the key points to understand:

- The `SELECT` statement is structured as follows: `SELECT column_name(s) FROM table_name`.
- You can use the asterisk (*) character to select all columns from a table.
- After executing the `SELECT` statement, you can fetch the result using functions like `mysqli_query()` and `mysqli_fetch_assoc()`.

Now let's see an example code snippet that demonstrates how to select data from the "employee" table and display the results:

A screenshot of a code editor window titled 'php' with a 'Copy code' button in the top right corner. The code is a PHP script that connects to a MySQL database, executes a SELECT query, and displays the results. The code is as follows:

```
<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// SQL query to select data
$sql = "SELECT id, firstname, lastname FROM employee";
$result = mysqli_query($conn, $sql);

// Check if any rows are returned
if (mysqli_num_rows($result) > 0) {
    // Loop through each row of the result set
    while ($row = mysqli_fetch_assoc($result)) {
        echo "id: " . $row["id"] . " - Name: " . $row["firstname"] . " " . $row["lastname"] . "<br>";
    }
} else {
    echo "0 results";
}

// Close the connection
mysqli_close($conn);
?>
```

In the above code, replace "your_username", "your_password", and "your_database" with your actual MySQL credentials and the name of your database.

The code establishes a connection to the MySQL server using `mysqli_connect`, and if the connection is successful, it executes the SQL query to select the "id", "firstname", and "lastname" columns from the "employee" table. The result is stored in the `$result` variable.

We then check if any rows are returned using `mysqli_num_rows`, and if there are results, we iterate over each row using `mysqli_fetch_assoc` and display the desired information.

If no rows are returned, the message "0 results" is displayed. Finally, don't forget to close the connection using `mysqli_close` to free up resources after you're done with the database operations.

UPDATE STATEMENT

Once you have established a connection to the MySQL database, you can use the `UPDATE` statement to modify existing records in a table. The `UPDATE` statement allows you to specify the columns you want to update and the new values for those columns. You can also use the `WHERE`

clause to specify which records should be updated based on certain conditions.

Here are the key points to understand:

- The UPDATE statement is structured as follows: UPDATE table_name SET column1=value1, column2=value2, ... WHERE some_column=some_value.
- The SET keyword is used to assign new values to the specified columns.
- The WHERE clause specifies the condition that determines which records should be updated. If you omit the WHERE clause, all records in the table will be updated.

Now let's see an example code snippet that demonstrates how to update a record in the "Employee" table:

```
php
Copy code

<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// SQL query to update data
$sql = "UPDATE Employee SET lastname='Astegnachew' WHERE id=11";

if (mysqli_query($conn, $sql)) {
    echo "Record updated successfully";
} else {
    echo "Error updating record: " . mysqli_error($conn);
}

// Close the connection
mysqli_close($conn);
?>
```

In the above code, replace "your_username", "your_password", and "your_database" with your actual MySQL credentials and the name of your database.

The code establishes a connection to the MySQL server using `mysqli_connect`, and if the connection is successful, it executes the SQL query to update the "lastname" column in the

"Employee" table. It specifies the new value for the "lastname" column as "Astegnachew" for the record with an "id" of 11.

If the update is successful, it displays the message "Record updated successfully". Otherwise, it displays an error message along with the MySQL error information. Finally, don't forget to close the connection using `mysqli_close` to free up resources after you're done with the database operations.

DELETE STATEMENT

Once you have established a connection to the MySQL database, you can use the DELETE statement to remove records from a table. The DELETE statement allows you to specify a condition using the WHERE clause to determine which records should be deleted. If you omit the WHERE clause, all records in the table will be deleted.

Here are the key points to understand:

- The DELETE statement is structured as follows: DELETE FROM table_name WHERE some_column = some_value.
- The FROM keyword specifies the table from which records should be deleted.
- The WHERE clause specifies the condition that determines which records should be deleted. If you omit the WHERE clause, all records in the table will be deleted.

Now let's see an example code snippet that demonstrates how to delete a record from the "Employee" table:

```

php
Copy code

<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// SQL query to delete a record
$sql = "DELETE FROM Employee WHERE id=6";

if (mysqli_query($conn, $sql)) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . mysqli_error($conn);
}

// Close the connection
mysqli_close($conn);
?>

```

In the above code, replace "your_username", "your_password", and "your_database" with your actual MySQL credentials and the name of your database.

The code establishes a connection to the MySQL server using `mysqli_connect`, and if the connection is successful, it executes the SQL query to delete the record from the "Employee" table where the "id" is 6.

If the deletion is successful, it displays the message "Record deleted successfully". Otherwise, it displays an error message along with the MySQL error information. Finally, don't forget to close the connection using `mysqli_close` to free up resources after you're done with the database operations.

DEALING WITH FORMS IN PHP

In HTML forms, the method attribute specifies how the form data will be sent to the server. There are two commonly used methods: GET and POST.

GET Method:

- When the method attribute of a form is set to "GET", the form data is appended to the URL as query parameters.

- The form data is visible in the URL, which means it can be bookmarked and shared.
- GET requests are limited in the amount of data they can send. URLs have a maximum length, and the data should be kept within that limit.
- This method is suitable for forms that retrieve data or perform read operations.
- In PHP, the form data submitted via the GET method can be accessed using the `$_GET` superglobal array.

POST Method:

- When the method attribute of a form is set to "POST", the form data is sent as part of the HTTP request body.
- The form data is not visible in the URL, making it more secure and suitable for sending sensitive information like passwords.
- POST requests can handle large amounts of data as they are not limited by URL length restrictions.
- This method is suitable for forms that submit data and perform write operations.
- In PHP, the form data submitted via the POST method can be accessed using the `$_POST` superglobal array.
- In both cases, the form data can be accessed in PHP by referring to the name attribute of the form fields. For example, `$_GET['fieldname']` or `$_POST['fieldname']`.

It's important to consider the nature of the data being transmitted and the security requirements when choosing between the GET and POST methods in HTML forms. GET is commonly used for simple and non-sensitive data retrieval, while POST is suitable for transmitting larger data or sensitive information.

Dealing with forms in PHP involves handling user input from HTML forms and processing that data on the server side. PHP provides various functions and techniques to handle form data,

validate inputs, and perform necessary actions based on the submitted form data.

Here is a step-by-step explanation of how to deal with forms in PHP :-

HTML Form :-

- Start by creating an HTML form using the `<form>` element.
- Specify the form's method attribute as either "GET" or "POST" to determine how the form data will be sent to the server.
- Set the form's action attribute to specify the URL or PHP file that will process the form data.

Input Fields:

- Inside the `<form>` element, add input fields such as text fields, checkboxes, radio buttons, dropdown menus, etc., using appropriate HTML input elements like `<input>`, `<textarea>`, `<select>`, etc.
- Assign a unique name attribute to each input field. This name will be used to identify the form field when processing the form data in PHP.

Submit Button:

- Include a submit button inside the form using the `<input>` element with the type attribute set to "submit".
- The submit button allows the user to submit the form data to the server.

PHP Form Processing:

- Create a separate PHP file that will handle the form data once it is submitted.
- In the PHP file, you can access the form data using the `$_POST` or `$_GET` superglobal array, depending on the form's method attribute.
- Retrieve the values of form fields using the `$_POST['fieldname']` or `$_GET['fieldname']` syntax, where 'fieldname' is the name attribute of the input field.

- Validate and sanitize the form data to ensure it meets your desired criteria and is secure for further processing.
- Perform necessary actions based on the form data, such as storing it in a database, sending an email, generating a response, etc.
- You can also redirect the user to another page or display a success/failure message after processing the form.

Now, let's see an example of a simple HTML form and its corresponding PHP form processing code:

Now, let's see an example of a simple HTML form and its corresponding PHP form processing code:

HTML Form (index.html):

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
<title>PHP Form Example</title>
</head>
<body>
<h2>PHP Form Example</h2>
<form method="POST" action="process.php">
<label for="name">Name:</label>
<input type="text" id="name" name="name" required><br>

<label for="email">Email:</label>
<input type="email" id="email" name="email" required><br>

<input type="submit" value="Submit">
</form>
</body>
</html>
```

PHP Form Processing (process.php):

```
php
Copy code
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
// Retrieve form data
$name = $_POST['name'];
$email = $_POST['email'];

// Validate form data (e.g., check for required fields, email format, etc)
// Perform necessary actions with the form data (e.g., store in database,

// Display a success message
echo "Thank you, $name! Your form has been submitted successfully.";
}
?>
```

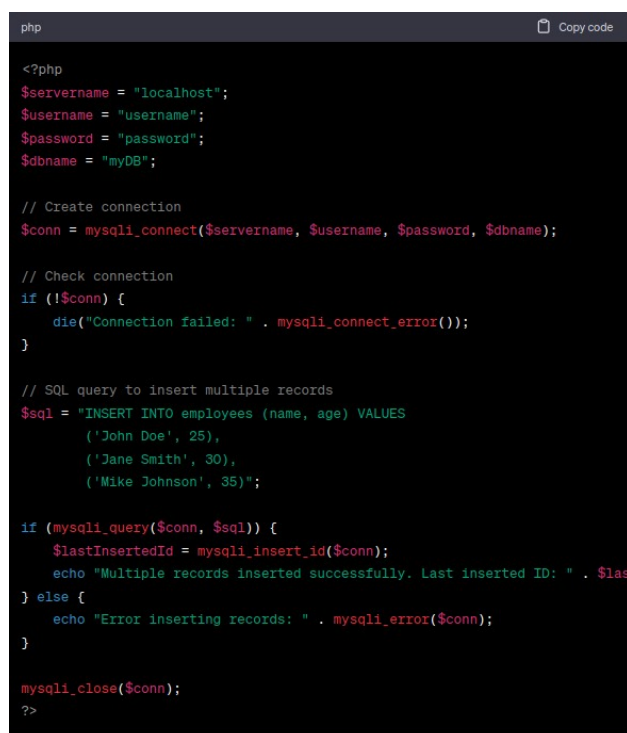
In the above example, when the user submits the form, the form data is sent to the "process.php" file using the POST method. In "process.php", the

form data is retrieved using the `$_POST` superglobal array.

Get the id of the last record item and insert multiple records

To get the ID of the last inserted record in PHP, you can use the `mysqli_insert_id()` function. This function retrieves the automatically generated ID that was used in the last query.

Here's an example that demonstrates inserting multiple records into a MySQL database using PHP and obtaining the ID of the last inserted record :-

A screenshot of a code editor window titled 'php' with a 'Copy code' button. The code is a PHP script that connects to a MySQL database, inserts three records into an 'employees' table, and displays the last inserted ID. The code is as follows:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// SQL query to insert multiple records
$sql = "INSERT INTO employees (name, age) VALUES
      ('John Doe', 25),
      ('Jane Smith', 30),
      ('Mike Johnson', 35)";

if (mysqli_query($conn, $sql)) {
    $lastInsertedId = mysqli_insert_id($conn);
    echo "Multiple records inserted successfully. Last inserted ID: " . $lastInsertedId;
} else {
    echo "Error inserting records: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

In this example, we establish a connection to the MySQL database and then execute an INSERT query to insert multiple records into the "employees" table. After the query is executed successfully, we retrieve the last inserted ID using the `mysqli_insert_id()` function and display it as output.

SELECT STATEMENT WITH HTML FORM

Using a SELECT statement with an HTML form allows users to input values and retrieve data from a database based on those inputs. It enables dynamic filtering and retrieval of specific data based on user preferences or search criteria.

Here's a brief overview of how it works:

1. **Create an HTML form:** Build an HTML form with input fields that allow users to enter their search criteria. For example, you can have text fields, dropdown menus, checkboxes, or radio buttons.
2. **Submit the form:** When the user submits the form by clicking a submit button, the form data is sent to a PHP script for processing.
3. **Process the form data:** In the PHP script, retrieve the form data using the `$_GET` or `$_POST` superglobal arrays, depending on the method used in the form (GET or POST). Store the form data in variables for further use.
4. **Construct the SELECT query:** Use the form data stored in variables to construct a dynamic SELECT query that retrieves data from the database based on the user's input. You can use conditions, filters, and placeholders in the query to ensure proper data retrieval.
5. **Execute the SELECT query:** Execute the SELECT query using a database connection and fetch the results.
6. **Display the results:** Use PHP to process and display the retrieved data on the webpage. This can be done by iterating over the result set and formatting the data in a suitable way, such as in a table or list.

Here's an example that demonstrates a simple HTML form with a SELECT statement in PHP:

```
html Copy code
<!DOCTYPE html>
<html>
<head>
  <title>Search Form</title>
</head>
<body>
  <form action="search.php" method="POST">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name">

    <label for="category">Category:</label>
    <select id="category" name="category">
      <option value="books">Books</option>
      <option value="electronics">Electronics</option>
      <option value="clothing">Clothing</option>
    </select>

    <input type="submit" value="Search">
  </form>
</body>
</html>
```

```
php Copy code
<?php
// Check if the form is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  // Retrieve form data
  $name = $_POST["name"];
  $category = $_POST["category"];

  // Validate and sanitize the inputs (you can add your own validation logic)

  // Create a database connection
  $servername = "localhost";
  $username = "your_username";
  $password = "your_password";
  $dbname = "your_database";
  $conn = mysqli_connect($servername, $username, $password, $dbname);

  // Check the connection
  if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
  }

  // Construct the SELECT query
  $query = "SELECT * FROM products WHERE name LIKE '%$name%' AND category";

  // Execute the query
  $result = mysqli_query($conn, $query);

  // Check if any rows are returned
  if (mysqli_num_rows($result) > 0) {
    // Display the results
    echo "<table>";
    echo "<tr><th>Name</th><th>Category</th><th>Price</th></tr>";
    while ($row = mysqli_fetch_assoc($result)) {
      echo "<tr>";
      echo "<td>" . $row["name"] . "</td>";
      echo "<td>" . $row["category"] . "</td>";
      echo "<td>" . $row["price"] . "</td>";
      echo "</tr>";
    }
    echo "</table>";
  } else {
    echo "No results found.";
  }

  // Close the database connection
  mysqli_close($conn);
}
```

The form above contains two input fields: "Name" (a text field) and "Category" (a dropdown menu). The form's action is set to "search.php", which is the PHP script that will process the form data and execute the SELECT query.

In the "search.php" file, you would handle the form submission and construct the SELECT query based on the submitted values. You can access the form data using `$_POST['name']` and `$_POST['category']` to retrieve the user's inputs. Then, you can use these values in your SELECT query to filter and retrieve data from the database accordingly.

Here's an example of a PHP script that handles the form submission and performs a SELECT query based on the user's input:

In this example, we assume that you have a table named "products" in your database with columns "name", "category", and "price". Adjust the table and column names according to your database structure.

When the form is submitted, the PHP script retrieves the form data (\$name and \$category), validates and sanitizes the inputs (not shown in this example), creates a database connection, constructs the SELECT query with the user's inputs, executes the query, and displays the results in an HTML table.

DELETE STATEMENT WITH HTML FORM

Here's an example of an HTML form (delete.html) that allows the user to input the ID of the record they want to delete :-


```

html
Copy code
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Delete Record</title>
</head>
<body>
  <h2>Delete Record</h2>
  <form action="delete.php" method="POST">
    <label for="product_id">Product ID:</label>
    <input type="text" name="product_id" id="product_id" required>
    <br>
    <input type="submit" value="Delete">
  </form>
</body>
</html>

```

In this HTML form, we have a text input field where the user can enter the ID of the record they want to delete. The form has an action attribute set to delete.php, which is the PHP script that will handle the form submission and perform the delete operation.

Make sure to save this HTML code in a file named delete.html. When the form is submitted, it will send the entered product ID to the delete.php script for further processing.

Here's an example of a PHP script that handles the form submission and performs a DELETE query based on the user's input :-

```

php
Copy code
<?php
// Check if the form is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Retrieve form data
    $productID = $_POST["product_id"];

    // Validate and sanitize the input (you can add your own validation logic)

    // Create a database connection
    $servername = "localhost";
    $username = "your_username";
    $password = "your_password";
    $dbname = "your_database";
    $conn = mysqli_connect($servername, $username, $password, $dbname);

    // Check the connection
    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }

    // Construct the DELETE query
    $query = "DELETE FROM products WHERE id = $productID";

    // Execute the query
    if (mysqli_query($conn, $query)) {
        echo "Record deleted successfully.";
    } else {
        echo "Error deleting record: " . mysqli_error($conn);
    }

    // Close the database connection
    mysqli_close($conn);
}
?>

```

In this example, we assume that you have a table named "products" in your database with a column named "id" as the primary key. Adjust the table and column names according to your database structure.

When the form is submitted, the PHP script retrieves the form data (\$productID), validates and sanitizes the input (not shown in this example), creates a database connection, constructs the DELETE query using the user's input, executes the query, and displays the success message or the error message, if any.

UPDATE STATEMENT WITH HTML FORM

Here's an example of an HTML form (update.html) that allows the user to update a record by entering the product ID and the new quantity:

```

html
Copy code
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Update Record</title>
</head>
<body>
  <h2>Update Record</h2>
  <form action="update.php" method="POST">
    <label for="product_id">Product ID:</label>
    <input type="text" name="product_id" id="product_id" required>
    <br>
    <label for="quantity">New Quantity:</label>
    <input type="number" name="quantity" id="quantity" required>
    <br>
    <input type="submit" value="Update">
  </form>
</body>
</html>

```

In this HTML form, we have two input fields. The first one is for the product ID, which is used

to identify the record to update. The second field is for the new quantity that will be updated.

The form has an action attribute set to update.php, which is the PHP script that will handle the form submission and perform the update operation.

Make sure to save this HTML code in a file named update.html. When the form is submitted, it will send the entered product ID and new quantity to the update.php script for further processing.

```
php
Copy code

<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Check if the form is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Get the values from the form
    $product_id = $_POST["product_id"];
    $quantity = $_POST["quantity"];

    // Update the record in the database
    $sql = "UPDATE products SET quantity = '$quantity' WHERE id = '$product_id'";

    if (mysqli_query($conn, $sql)) {
        echo "Record updated successfully";
    } else {
        echo "Error updating record: " . mysqli_error($conn);
    }
}

// Close the database connection
mysqli_close($conn);
?>
```

In this PHP script, we first establish a connection to the database using the provided credentials. Then, we check if the form is submitted using `$_SERVER["REQUEST_METHOD"]`. If it is, we retrieve the values of `product_id` and `quantity` from the `$_POST` superglobal.

Next, we construct an SQL query to update the products table with the new quantity for the specified product ID. We use the `mysqli_query()` function to execute the query. If the update is successful, we display a success message. Otherwise, we display an error message along with the error details.

Finally, we close the database connection using `mysqli_close()`.

Make sure to save this PHP code in a file named update.php and ensure that you have the correct database credentials and table structure in place.

SESSIONS IN PHP ?

Sessions in PHP allow you to store and retrieve data across multiple requests from the same user. A session is a way to keep track of user information as they navigate through your website or web application. It enables you to store user-specific data, such as login credentials, shopping cart contents, or user preferences.

```
html
Copy code

<!DOCTYPE html>
<html>
<head>
    <title>Delete Record</title>
</head>
<body>
    <form action="delete.php" method="POST">
        <label for="id">ID:</label>
        <input type="text" name="id" id="id">
        <input type="submit" value="Delete">
    </form>
</body>
</html>

delete.php:

php
Copy code

<?php
session_start();

// Retrieve the ID from the session
$id = $_SESSION['id'];

// Database connection code
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "yourdatabase";

$conn = mysqli_connect($servername, $username, $password, $dbname);
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Construct the DELETE query
$sql = "DELETE FROM your_table WHERE id = '$id'";

// Execute the DELETE query
if (mysqli_query($conn, $sql)) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

In this example, the delete.html file contains an HTML form that submits the ID to the delete.php script. The ID value is then retrieved from the session in delete.php. The script establishes a

database connection and constructs the DELETE query using the retrieved ID. Finally, the script executes the DELETE query to delete the record from the database.