**ASSIGNMENT**

**HTML Answers**

1. <!DOCTYPE html> is it a tag of html? If not, what is it and why do we use it?

* <DOCTYPE html> is not a tag in html, it is known as “**Document Type Declaration**” or “**Document Type Definition**” (DTD).
* It is used to define the type or version of html being used in webpage.
* <DOCTYPE html> declaration is used in html to indicate that the document is written in HTML syntax.
* The purpose of <DOCTYPE html> is to inform the browser about the version of html document, which helps the browser to render the page correctly.
* It helps to ensure better cross-browser compatibility and consistent rendering of webpages across different devices and platforms.

1. Explain Semantic tags in html? And why do we need it?

* Semantic tags in HTML are elements that provide meaning and structure to the content within a webpage.
* Unlike generic tags **“<div>”** and **“<span>”**. Semantic tags describe the purpose and role of the content they wrap.
* They help both humans and search engines to understand the significance of different sections of a webpage.

Here are some examples of semantic tags introduced in HTML5: -

**<header>,<nav>, <main>,<article>,<section>,<aside>, <footer>**

In summary, semantic tags in HTML add meaning and structure to webpage content improving accessibility, search engine optimization and forward compatibility.

1. Differentiate between HTML Tags and Elements?

**Tags –** Tags are fundamental building blocks of HTML syntax. They are used to define the structure and appearance of elements within HTML document. Tags are represented by angle brackets **“<”** and **“>”** and usually comes in pairs with an opening tag and closing tag.

Ex-

**<p> ………. </p>** is an example of tag. **<p>** is the known as opening tag and **</p>** is known as closing tag.

It is important to note that some tags are void or empty don’t having closing tag and are self-closed with forward slash **( ‘/ ‘)** before the closing angle bracket of the opening tag for example,

<img src=”image.png” alt=”image” />, <br/> and <hr/> etc.

**Elements –** Elements are composed of tags, along with the content or other element they enclose. An element consists of an opening tag with content and a closing tag. The opening tag marks the beginning of the element and closing tag marks the end of the element. The content be text or other elements or a combination of both.

Ex –

**<p> This is paragraph</p>** as a whole is an example of element where **<p>** is opening tag **“This is paragraph”** is content and **</p>** is closing tag.

1. Build Your Resume using HTML only.

Repo Link - <https://github.com/Gaurav-Dev24/Placement-Assignment/blob/main/HTML%20Answers/Q.no_4_resume.html>

1. Write Html code so that it looks like the given image Link

Repo Link - <https://github.com/Gaurav-Dev24/Placement-Assignment/blob/main/HTML%20Answers/Q.no_5_image.html>

1. What are some of the advantages of HTML5 over its previous versions?

HTML5 is the fifth revision of Hyper Text Mark-up Language, which has been introduced with several significant advancements over previous versions. Some of the major changes are as followed:-

* **Rich Multimedia Support**
* **Improved Backward Compatibility**
* **Enhanced New Semantics such as <header>, <footer>, <article>, <section>**
* **Improved form elements**
* **Introduction of localStorage API**

1. Create a simple Music player using html only

Repo Link - <https://github.com/Gaurav-Dev24/Placement-Assignment/blob/main/HTML%20Answers/Q.no_7_music_player.html>

1. What is the difference between <figure> tag and <img> tag?

The major difference between <img> and <figure> is <img> tag is used to embed image in html document. It is self-closing tag that require src attributes to specify the image source or URL.

Whereas <figure> tag is used to group and represent self-contained content such as images, diagrams and code snippets. It provides a way to associate caption or description with the content it encloses.

Apart from it <img> tag is primarily used for displaying image it does not have any additional semantic meaning whereas in <figure> tag, it provides semantic meaning to the content.

Syntax of image tag - <img src=”./images/img.png” alt=”image”/>

Syntax for figure tag –

<figure>

<img src=”./images/img.png” alt=”image”/>

<figcaption>Caption</figcaption>

</figure>

1. What’s the difference between html tag and attribute and give example of some global attributes?

Html Tag- An HTML tag represents an element in an HTML document and is enclosed within angle brackets. Tags define the structure and content of the document. They can be categorising into two types “Opening tag” and “Closing tag”. The opening tag denotes the start and closing tag denotes the end of the element. Some elements known as “void” and “empty” does not require closing tag hence known as self-closing element.

Some examples of tags are:

**<p>This is paragraph tag</p>**

**<a href=”#” >This is anchor tag</a**>

<img src=”./images/img.png” alt=”image”/>

Attributes- An attribute is provided in HTML tag to modify or to provide additional information about the element. Attributes are placed inside the opening tag and consist of name-value pair. The attribute name is followed by “=” sign and attribute value is followed by “ “ and ‘ ‘ quotes.

Some of the examples of attributes are;

href, type, target, placeholder src, alt, width and height etc.

<a href="https://www.example.com" target="\_blank">Link</a>

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

<img src="image.jpg" alt="Image description" width="500" height="300">

Examples of global attributes are:

* Class – specifies one or more classes to an element
* Id – provides unique identifier to the element
* Style – defines inline css style to the element
* Title – specifies the title or tooltip text for element
* Lang – defines the language of the elements content
* Data – allows custom data attribute to be added to element

1. Build Table which looks like the given image Link

Repo Link - <https://github.com/Gaurav-Dev24/Placement-Assignment/blob/main/HTML%20Answers/Q.no_10_table.html>

**CSS Answers**

1. What is Box Model in CSS & Which CSS Properties is part of it?

The Box model in css describes the layout and sizing of elements on a web page. It consist of four components which defines the total space occupied by an element: content, padding, border, margin.

The properties associated with box model are:

Content –

* Width – specifies the width of the content area
* Height – specifies the height of the content area

Padding –

* top, right, bottom and left determines the space between the content area and border of element.

Border –

* border-width – sets the thickness of border
* border-style – specifies the style of the border (e.g., solid, dashed etc.)
* border-color – specifies the color of the border
* border-radius – specifies the rounded corners of the border.

Margin –

* top, right, bottom and left determines the space between elements border and neighbouring elements.

Together all these properties controls the overall size and spacing of elements in web page.

e.g.-

.box {

width: 200px;

height: 150px;

padding: 10px;

border: 1px solid black;

margin: 20px;

}

1. What are the Different Types of Selectors in CSS & what are the advantages of them?

There are different types of selectors present in CSS to target specific HTML elements and applying css to them. Some of the commonly used selectors are as followed:

**Element Selector** – selects elements based on their element name.

**Eg.**

p {

/\* Styles applied to all <p> elements \*/

}

**Advantages** – simple and straight forward, targets all the element of specific type.

**Class Selectors** – selects elements based on the value of their “class” attribute. Starts with (.) followed by name of the class.

**Eg**.

.highlight {

/\* Styles applied to elements with class="highlight" \*/

}

**Advantages** – allows the grouping and styling of multiple elements across different tags with same class name promoting reusability and maintaining consistency.

**ID Selectors** – selects elements based on the unique value of their “id” attribute. Starts with (#) followed by name of the id.

**Eg**.

. #header {

/\* Styles applied to the element with id="header" \*/

}

**Advantages** – target a single unique element, useful for applying specific styles or JavaScript interactions.

**Attribute Selectors** – selects elements based on their attribute value.

**Eg**.

input[type="text"] {

/\* Styles applied to <input> elements with type="text" \*/

}

**Advantages** –allows targeting element based on the value of attribute providing flexibility and granularity in slelction.

**Descendent Selectors** – selects elements that are descendants of another element.

**Eg**.

.container p {

/\* Styles applied to <p> elements within an element with class="container" \*/

}

**Advantages** –enables targeting specific elements within a particular parent container, facilitating contextual styling.

**Child Selectors** – selects direct child of an element.

**Eg**.

ul > li {

/\* Styles applied to <li> elements that are direct children of <ul> \*/

}

**Advantages** –targets immediate child elements, useful for styling the specific level of nested elements.

**Pseudo-classes and Pseudo-elements Selectors** – select the elements based on their state or position.

**Eg**.

a:hover {

/\* Styles applied to <a> elements on hover \*/

}

p::first-letter {

/\* Styles applied to the first letter of <p> elements \*/

}

**Advantages** – enables styling based on dynamic states (e.g. hover, active) or applying styles to specific parts of an element (e.g. first letter, first line).

1. What is VW/VH & How it’s different from PX?

VW (Viewport Width) and VH (Viewport Height) are units of measurement in CSS that are relative to the size of viewport or the browser window.

VW – 1vw = 1% of the width of viewport.

e.g.

If the width of the viewport is 1000px then 1vw is equal to 10 pixels i.e. 1% \* 1000px.

VH – 1vh = 1% of the height of viewport.

e.g.

If the height of the viewport is 800px then 1vw is equal to 8 pixels i.e. 1% \* 800px.

In summary the major difference between the VW/VH and PX is how they are calculated and their responsiveness o changes in viewport size. VW/VH are particularly useful for creating responsive designs that dynamically adapt to the different viewport size, whereas PX provide a fixed size regardless the viewport dimensions.

1. What’s difference between Inline, Inline Block and block?

The difference between “inline”, “inline-block” and “block” in css determines that how elements are rendered and how they interact with other elements within the document flow.

**Inline:** Inline elements do not start on a new line and only take up as much width as necessary to contain their content. They flow alongside each other horizontally within the same line.

Examples of inline elements: <span>, <a>, <strong>, <em>, <img>.

**Inline Block:** Inline-block elements are similar to inline elements in that they flow within the same line horizontally, but they can have a specified width, height, margins, and padding. They retain their block-level properties while staying inline.

Examples of inline-block elements: <input>, <button>, <select>, <div> (with display: inline-block;).

**Block:** Block-level elements start on a new line and take up the full width available, extending from the left edge to the right edge of their containing element by default. They create a block-level formatting context and can contain other block-level and inline elements.

Examples of block-level elements: <div>, <p>, <h1>-<h6>, <ul>, <li>, <section>, <header>, <footer>, etc.

1. How is Border-box different from Content Box?

Content Box:

* The "content-box" is the default box-sizing value in CSS.
* When using "content-box", the specified width and height of an element only include the content area, excluding the padding, border, and margins.
* In other words, the width and height you set for an element using "content-box" represent the dimensions of the content area alone, and the padding, border, and margins are added to it.

Border Box:

* When using "border-box", the specified width and height of an element include the content area, padding, and border.
* In "border-box" sizing, the padding and border are included within the specified width and height, and the browser adjusts the content box size to fit.

The choice between "content-box" and "border-box" depends on the desired box-sizing behaviour. "border-box" is often favoured for its convenience, as it allows easier control over the overall dimensions of elements, especially when dealing with complex layouts and responsive designs.

1. What’s z-index and how does it function?

"z-index" property appropriately allows developers to control the layering and visibility of elements on the web page. It is commonly used in scenarios where elements need to overlap or when creating complex layouts with overlapping components, dropdown menus, or modal dialogs. It's important to note that the "z-index" property affects only elements that have a position value other than "static".

1. What’s Grid & Flex and difference between them?

Grid and Flex box are two popular CSS layout systems that provide different approaches to creating responsive and flexible web page layouts.

**Flex box (Flexible Box Layout):**

Flex box is a one-dimensional layout system that allows you to arrange elements along a single axis (either horizontally or vertically). It provides powerful alignment and distribution capabilities for creating flexible and dynamic layouts.

**Key points about Flex box include:** one dimensional layout, flex containers& flex items, flexibility & alignment.

**Grid:**

Grid is a two-dimensional layout system that allows you to create complex grid-based layouts with rows and columns. It provides precise control over both horizontal and vertical axes, enabling you to create responsive layouts that adapt to different screen sizes.

**Key points about Grid include:** two dimensional layouts, grid containers & grid items, rows & columns and grid tracks & grid areas

Difference between Grid and Flex box:

1. **Layout system:** Flex box is a one-dimensional layout system, while Grid is a two-dimensional layout system.
2. **Axis control:** Flex box works along a single axis (horizontal or vertical), whereas Grid allows control over both the horizontal and vertical axes.
3. **Alignment:** Flex box provides powerful alignment properties for flex items, allowing you to control their positioning within a flex container. Grid offers more advanced alignment capabilities, including alignment of both rows and columns.
4. **Layout complexity:** Grid is more suitable for complex grid-based layouts with multiple rows and columns, whereas Flex box is well-suited for simpler one-dimensional layouts or aligning items within a container.
5. Difference between absolute and relative and sticky and fixed position explain with example.

The differences between absolute and relative positioning, as well as sticky and fixed positioning, are related to how elements are positioned within the document flow and their behaviour when scrolling.

**Absolute Positioning:**

* Absolute positioning removes an element from the normal document flow and positions it relative to its nearest positioned ancestor or the containing block.
* The element's position is specified using the **top, right, bottom,** and **left** properties.
* Other elements are not affected by the absolute positioned element, and it may overlap or be overlapped by other elements.
* If no positioned ancestor is found, the element will be positioned relative to the initial containing block, which is usually the viewport.

e.g.

**<div class="container">**

**<div class="absolute-box"></div>**

**</div>**

**.container {**

**position: relative;**

**height: 300px;**

**}**

**.absolute-box {**

**position: absolute;**

**top: 50px;**

**left: 50px;**

**width: 200px;**

**height: 200px;**

**background-color: red;**

**}**

In the above example, the **.absolute-box** element is positioned absolutely within its relative positioned parent container. It is placed **50 pixels** from the top and **50 pixels** from the left of its nearest positioned ancestor.

**Relative Positioning:**

* Relative positioning allows an element to be positioned relative to its normal position in the document flow.
* The element's position is adjusted using the **top, right, bottom,** and **left** properties, which shift it from its original position.
* Other elements on the page are not affected by the relative positioned element, and it still occupies its original space in the document flow.

e.g.

**<div class="relative-box"></div>**

**.relative-box {**

**position: relative;**

**top: 20px;**

**left: 30px;**

**width: 150px;**

**height: 150px;**

**background-color: blue;**

**}**

In the above example, the **.relative-box** element is positioned relative to its original position. It is shifted **20 pixels down** and **30 pixels to the right** from where it would normally appear.

**Sticky Positioning:**

* Sticky positioning is a combination of both relative and fixed positioning.
* It behaves like a relatively positioned element until it reaches a specific scroll position, after which it becomes fixed in place.
* The element's position is determined using the **top, right, bottom,** and **left** properties when it becomes sticky.
* Sticky positioning is commonly used for creating sticky headers, navigation menus, or sidebars that stay fixed until a certain point while scrolling.

e.g.

**<div class="sticky-box"></div>**

**.sticky-box {**

**position: sticky;**

**top: 20px;**

**width: 200px;**

**height: 100px;**

**background-color: green;**

**}**

In the above example, the **.sticky-box** element starts as a relatively positioned element. However, when the user scrolls to a specific point (defined by **top: 20px;**), it becomes fixed in place and remains visible even as the rest of the content scrolls.

**Fixed Positioning:**

* Fixed positioning removes an element from the normal document flow and positions it relative to the viewport (the browser window).
* The element's position is specified using