**Q.1 What’s Box Model in CSS ?**

The Box Model is a significant concept in CSS that specifies how items on a web page are structured and displayed. It specifies an element's characteristics and layout, such as its content, padding, border, and margin.

The Box Model is made up of four major components:

**The content box** contains the content of an element, such as text, photos, or other HTML components. The width and height attributes determine the size of the content box.

**Padding** is the space between the content and the border of an element. It offers spacing and is adjustable using the padding property. Padding can be specified separately for each side (top, right, bottom, and left) or using shorthand notation.

**Border**: The border surrounds the padding and content boxes, defining the element's visible boundary. The border property allows you to change the size, style, and color of the border.

**Margin**: The space outside the element that creates a gap between it and other elements is known as the margin. It is used for spacing and is controlled by the margin property. Margins, like padding, can be specified individually or using shorthand notation.

The overall size of an element is calculated by summing the width and height of the content, padding, border, and margin. For example, if an element has a width of 200 pixels, 10 pixels of padding, 1 pixel of border, and a margin of 20 pixels, the overall width of the element is 242 pixels (200 + 10 + 1 + 20 + 10 + 1).

**Q.2 What are the Different Types of Selectors in CSS & what are the advantages of them?**

CSS provides various types of selectors that allow you to target specific elements or groups of elements on a web page. Here are some commonly used selectors and their advantages:

1. Element Selector: This selects elements based on their HTML tag name. For example, `p` selects all `<p>` elements. Advantages:

- Easy to use and understand.

- Applies styles to all instances of the selected element.

2. Class Selector: This selects elements based on the value of their `class` attribute. For example, `.btn` selects all elements with the `class="btn"`. Advantages:

- Allows you to apply styles to multiple elements with the same class.

- Promotes code reusability and consistency.

- Works well with JavaScript for adding/removing classes dynamically.

3. ID Selector: This selects a single element based on the value of its `id` attribute. For example, `#header` selects the element with `id="header"`. Advantages:

- Targets a specific element.

- Useful for styling unique elements or applying specific behaviors with JavaScript.

4. Attribute Selector: This selects elements based on the presence or value of their attributes. For example, `[type="text"]` selects all elements with `type="text"`. Advantages:

- Allows targeting elements based on specific attribute values.

- Helpful for styling specific input types or elements with certain attributes.

5. Descendant Selector: This selects elements that are descendants of another element. For example, `ul li` selects all `<li>` elements that are descendants of a `<ul>` element. Advantages:

- Allows styling of specific elements within a nested structure.

- Provides flexibility in targeting elements at different levels.

6. Pseudo-classes and Pseudo-elements: Pseudo-classes select elements based on a specific state or condition, such as `:hover` for styling when hovering over an element. Pseudo-elements select and style a specific part of an element, such as `::before` to insert content before an element. Advantages:

- Allows styling based on user interaction or dynamic states.

- Enhances visual effects and interactivity.

Using these different types of selectors, you can precisely target elements and apply styles to achieve the desired appearance and behavior on your web pages. They provide flexibility, reusability, and specificity in styling, making your CSS more efficient and maintainable.

**Q.3 What is VW/VH ?**

`vw` and `vh` are units of measurement in CSS that represent the viewport width and viewport height, respectively. They are relative units that allow you to size elements based on a percentage of the viewport dimensions.

- Viewport Width (`vw`): `1vw` is equal to 1% of the viewport width. For example, if the viewport width is 1000 pixels, `1vw` would be equal to 10 pixels. The value of `vw` adjusts dynamically as the viewport size changes.

- Viewport Height (`vh`): `1vh` is equal to 1% of the viewport height. For instance, if the viewport height is 800 pixels, `1vh` would be equal to 8 pixels. The value of `vh` also adapts dynamically as the viewport size changes.

**Q.4 Whats difference between Inline, Inline Block and block ?**

In CSS, the display property determines how an element is rendered and how it interacts with other elements on the page. Here are the differences between the `inline`, `inline-block`, and `block` display properties:

1. \*\*Inline\*\*: Elements with `display: inline` are rendered inline, meaning they flow within the text content and do not create line breaks before or after themselves. They respect the surrounding text and other inline elements. Key characteristics of inline elements include:

- They do not have a width or height property.

- Margins and paddings on all sides are not applied horizontally (left and right) but can be applied vertically (top and bottom).

- They do not start on a new line and are rendered next to each other.

Examples of inline elements include `<span>`, `<a>`, `<strong>`, and `<em>`. By default, most HTML elements like `<b>`, `<i>`, and `<img>` are also inline elements.

2. \*\*Inline Block\*\*: Elements with `display: inline-block` share characteristics of both `inline` and `block` elements. They are rendered inline like `inline` elements, allowing other elements to be rendered on the same line. However, they can have a width and height, and margins and paddings are applied as expected on all sides. Key characteristics of inline-block elements include:

- They can have a specified width and height.

- Margins and paddings on all sides are applied as expected.

- They do not start on a new line by default.

This display value is useful when you want elements to be rendered inline but have block-like properties, such as applying width and height or adding margins and paddings.

3. \*\*Block\*\*: Elements with `display: block` are rendered as block-level elements. They create line breaks before and after themselves, taking up the full width available by default. Key characteristics of block elements include:

- They start on a new line and occupy the entire width available.

- They can have a specified width and height.

- Margins and paddings on all sides are applied as expected.

Examples of block-level elements include `<div>`, `<p>`, `<h1>`-`<h6>`, `<ul>`, `<li>`, and `<section>`. These elements are typically used to create structure and define sections of a webpage.

Understanding the differences between `inline`, `inline-block`, and `block` is important for controlling the layout and behavior of elements on your webpage. You can choose the appropriate display value based on your design requirements and desired rendering behavior.

**Q.5 How is Border-box different from Content Box?**

In CSS, the `box-sizing` property determines how the width and height of an element are calculated, which affects the overall size of the element. There are two main values for the `box-sizing` property: `content-box` and `border-box`. Here's how they differ:

1. Content Box (default): When `box-sizing: content-box` is applied to an element, the width and height properties define only the content area of the element. This means that any padding or border added to the element will increase its overall size beyond the specified width and height. The content box size does not include padding or border.

2. Border Box: When `box-sizing: border-box` is applied to an element, the width and height properties include the content, padding, and border areas within the specified values. The element's total width and height are calculated by including the padding and border within the specified dimensions. In other words, the size of the border box is the sum of the content, padding, and border.

**Q.6 What’s z-index and How does it Function ?**

The z-index property in CSS controls the vertical stacking order of elements that overlap. In other words, it determines which element appears as if it is physically closer to you. The z-index property only affects elements that have a position value other than static (the default).

The z-index property can be used to make sure that certain elements always appear on top of others, even if they are nested within each other. For example, you could use the z-index property to make sure that a modal dialog always appears on top of the rest of the page content.

The z-index property is a numeric value, and higher values indicate a higher stacking order. So, an element with a z-index of 10 will appear on top of an element with a z-index of 5. If two elements have the same z-index value, the element that was positioned last in the HTML code will be shown on top.

Here is an example of how to use the z-index property:

Code snippet

.modal {

position: absolute;

top: 0;

left: 0;

width: 100%;

height: 100%;

background-color: black;

opacity: 0.5;

z-index: 10;

}

In this example, the modal element will always appear on top of the rest of the page content, even if it is nested within other elements. This is because the z-index property of the modal element is set to 10.

The z-index property is a powerful tool that can be used to control the stacking order of elements on a web page. By understanding how the z-index property works, you can create web pages that are visually appealing and easy to use.

**Q.7 What’s Grid & Flex and difference between them?**

Grid and Flex are two layout models in CSS that are used to arrange elements on a web page. They are both powerful tools, but they have different strengths and weaknesses.

Grid is a two-dimensional layout model that allows you to create complex layouts with rows and columns. It is well-suited for layouts that need to be responsive, as it can adapt to different screen sizes. Grid is also a good choice for layouts that need to be highly customizable, as you have a lot of control over the positioning of elements.

Flex is a one-dimensional layout model that is used to arrange elements in rows or columns. It is well-suited for layouts that need to be flexible, as elements can be resized and reordered without breaking the layout. Flex is also a good choice for layouts that need to be responsive, as it can adapt to different screen sizes.

**Q.8 Difference between absolute and relative and sticky and fixed position explain with example.**

Absolute: You would use absolute positioning to position an element that needs to be positioned independently of the surrounding elements. For example, you could use absolute positioning to position a floating button or a modal dialog.

* Relative: You would use relative positioning to position an element that needs to be offset from its normal position in the flow of the document. For example, you could use relative positioning to position a navigation bar that is always displayed at the top of the page, even when the user scrolls down.
* Sticky: You would use sticky positioning to position an element that needs to "stick" to the top of the viewport when the user scrolls down. For example, you could use sticky positioning to position a header that always remains visible at the top of the page, even when the user scrolls down.
* Fixed: You would use fixed positioning to position an element that needs to be positioned relative to the viewport and does not move when the page scrolls. For example, you could use fixed positioning to position a chat window or a navigation bar that always remains visible at the top of the page, even when the user scrolls down.