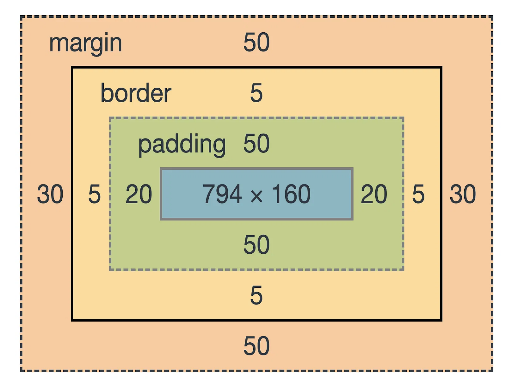
**iNeuron PPT Program Assignment – FJSJ Alpha Batch**

**CSS Questions**

**Q.1 Whats Box Model in CSS & Which CSS Properties are part of it ?**

The box model is a fundamental concept in CSS that describes how elements are rendered and how their content, padding, border, and margin are structured. The box model consists of four components:



1. **Content:** The actual content of the element, such as

text, images, or other nested elements.

1. **Padding:** The space between the content and the element's border. It can be set using the padding property.

1. **Border:** The line that surrounds the element's content and padding. It can be styled and customized using the border property.

1. **Margin:** The space between the element's border and neighboring elements. It can be set using the margin property.

These CSS properties are part of the box model:

* width and height: Specifies the width and height of the content area.

* padding: Sets the padding space around the content area.

* border: Defines the style, thickness, and color of the border.

* margin: Sets the space outside the border, creating the gap between elements.

By manipulating these properties, developers can control the layout, spacing, and sizing of elements on a webpage, enabling precise control over their appearance and positioning.

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

In CSS, there are several types of selectors that allow you to target and style specific elements or groups of elements. Here are some common types of CSS selectors and their advantages:

**1. Element Selectors:** These selectors target specific HTML elements based on their tag name. For example:

p {

  color: blue;

}

**2. Class Selectors:** Class selectors target elements based on the value of their "class" attribute. They are denoted by a dot (".") followed by the class name.

For example:

.highlight {

  background-color: yellow;

}

**3. ID Selectors:** ID selectors target elements based on the value of their "id" attribute. They are denoted by a hash ("#") followed by the ID name.

For example:

#navbar {

  font-size: 18px;

}

**4. Attribute Selectors:** Attribute selectors target elements based on the presence or value of their attributes. For example,

input[type="submit"] {

  background-color: green;

}

Here input[type="submit"] targets elements with the attribute "type" set to "submit".

**5. Pseudo-classes and Pseudo-elements:** Pseudo-classes and pseudo-elements target elements based on various conditions or specific parts of an element.

For example:

Pseudo-class Selector:

a:hover {

  text-decoration: underline;

}

This example targets <a> elements when the mouse hovers over them and adds an underline text decoration.

Pseudo-element Selector:

p::before {

  content: "Before ";

  font-weight: bold;

}

Here, a pseudo-element ::before is used to insert content before each <p> element with bold font weight.

The advantages of different CSS selectors include specificity, reusability, flexibility, and the ability to target elements based on various conditions. These selectors provide powerful tools for styling and manipulating the appearance of HTML elements in different ways, enhancing the overall design and user experience of web pages.

**Q.3 What is VW/VH & How its different from PX?**

VW (Viewport Width) and VH (Viewport Height) are units of measurement in CSS that allow you to specify dimensions relative to the size of the browser viewport. Here's how they differ from PX (pixels):

**VW (Viewport Width):**

VW represents a percentage of the width of the viewport.

1 VW is equal to 1/100th of the viewport width.

For example, width: 50vw; sets the width of an element to 50% of the viewport width.

**VH (Viewport Height):**

VH represents a percentage of the height of the viewport.

1 VH is equal to 1/100th of the viewport height.

For example, height: 75vh; sets the height of an element to 75% of the viewport height.

**PX (Pixels):**

PX is an absolute unit of measurement that represents the exact number of pixels.

It does not depend on the viewport size but rather on the device's screen resolution.

For example, width: 200px; sets the width of an element to exactly 200 pixels.

The main difference between VW/VH and PX is that VW and VH are relative units, based on the size of the viewport, while PX is an absolute unit based on the device's screen resolution. VW and VH allow for more responsive and fluid layouts, as the elements can scale proportionally with the viewport size. PX, on the other hand, provides a fixed and consistent size regardless of the viewport dimensions.

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

In CSS, there are three display properties that define how elements are rendered within the document flow: inline, inline-block, and block. Here's the difference between them:

**Inline:** Inline elements do not create line breaks and flow within the surrounding text.

They only take up the space necessary for their content and cannot have width and height specified. Examples of inline elements include <span>, <a>, and <strong>.

Inline elements can have vertical alignment and margin/padding applied, but the properties related to width and height are ignored.

**Inline-Block:** Inline-block elements share characteristics of both inline and block elements.

They flow within the surrounding text but can have width and height specified.

Inline-block elements respect padding and margin, and multiple inline-block elements can appear on the same line. Examples of inline-block elements include <img>, <input>, and <button>.

**Block:** Block elements create line breaks before and after themselves, taking up the entire width available.They can have width, height, padding, margin, and other box model properties specified. Block elements start on a new line and occupy the full width of their parent container.

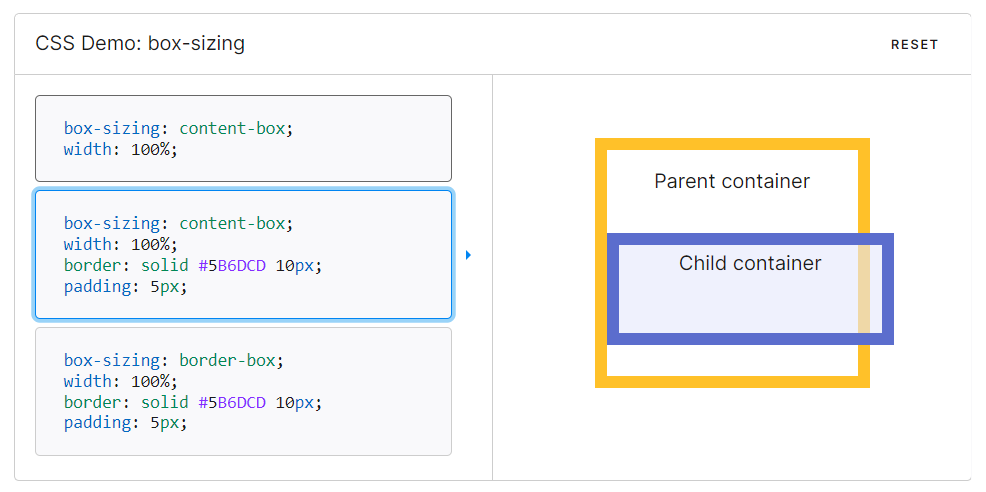
Examples of block elements include <div>, <p>, and <h1> to <h6>.

The main differences between inline, inline-block, and block elements lie in their positioning within the document flow, how they interact with surrounding elements, and their ability to have width and height specified. Inline elements flow within the text, inline-block elements flow within the text but can have specified width and height, and block elements create line breaks before and after themselves, taking up the full width available.

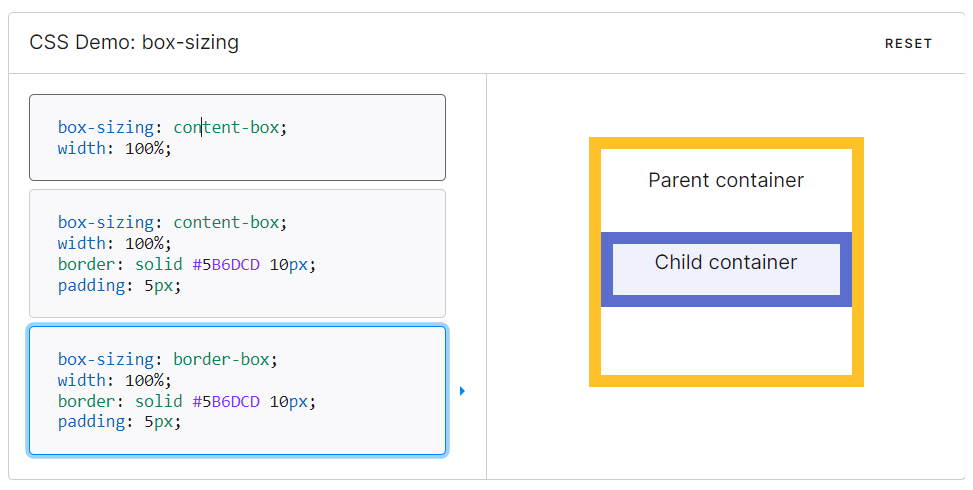
**Q.5 How is Border-box different from Content Box?**

The box-sizing CSS property sets how the total width and height of an element is calculated and it has two possible values: content-box and border-box. Here's how they differ:

**Content-box:** box-sizing: content-box gives you the default CSS box-sizing behavior. If you set an element's width to 100 pixels, then the element's content box will be 100 pixels wide, and the width of any border or padding will be added to the final rendered width, making the element wider than 100px.



**Border-box:** box-sizing: border-box tells the browser to account for any border and padding in the values you specify for an element's width and height. If you set an element's width to 100 pixels, that 100 pixels will include any border or padding you added, and the content box will shrink to absorb that extra width. This typically makes it much easier to size elements.



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

z-index is the CSS property that controls the stacking order of overlapping elements on a page. An element with a higher z-index value will appear in front of an element with a lower z-index value. The property is called “z-index” because it sets the order of elements along the z-axis. If the x-axis goes left-to-right and the y-axis goes top-to-bottom, the z-axis adds the dimension of “toward” and “away from” the user. Elements with a higher z-index value appear closer to the user, and elements with a lower value look farther away.

Function of z-index:

1. If “A” element has z-index: 1; and “B” element has z-index: 2; then B will be appear infront of A.

1. If two page elements overlap and no z-index value is given (or both elements have the same z-index value), the element placed last in the HTML code will appear on top of the element placed before it in the code.

1. Also, elements that are given a position value (other than static, the default value) appear above elements without a position in the stacking.

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

Here's an detailed comparision of Grid and Flexbox along with example code to illustrate their differences:

**Grid:** Grid is a two-dimensional layout system that creates a grid-based layout with rows and columns. It offers precise control over the placement and alignment of elements within the grid.

Grid is well-suited for creating complex grid structures and designing magazine-style layouts or evenly spaced grids.

Example code for a basic grid layout using CSS Grid:

<html>

<head>

<style>

.container {

  display: grid;

  grid-template-columns: 1fr 1fr 1fr;

  grid-gap: 10px;

}

.item {

  background-color: #f1f1f1;

  padding: 10px;

}

</style>

</head>

<body>

<div class="container">

  <div class="item">Item 1</div>

  <div class="item">Item 2</div>

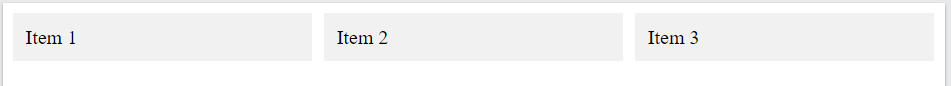
  <div class="item">Item 3</div>

</div>

</body>

</html>

**Output:**

****

In this example, a grid container is created using the display: grid; property. The grid-template-columns property defines three equal-width columns. The grid-gap property sets a 10px gap between grid items. Each grid item has a light gray background color and 10px padding.

**Flexbox:** Flexbox is a one-dimensional layout system that focuses on distributing space and aligning elements along a single axis.It is ideal for creating flexible and responsive layouts, vertically aligning elements, or creating flexible navigation menus.

Example code for a basic flexbox layout:

<html>

<head>

<style>

.container {

  display: flex;

  justify-content: space-between;

  align-items: center;

}

.item {

  background-color: #f1f1f1;

  padding: 10px;

}

</style>

</head>

<body>

<div class="container">

  <div class="item">Item 1</div>

  <div class="item">Item 2</div>

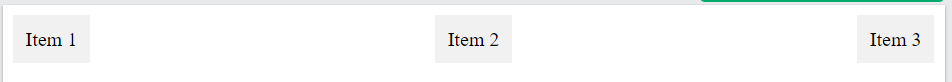
  <div class="item">Item 3</div>

</div>

</body>

</html>

**Output:**



In this example, a flex container is created using the display: flex; property. The justify-content property is set to space-between, which places the flex items at the beginning and end of the container, with equal spacing in between. The align-items property is set to center, which vertically aligns the flex items in the center. Each flex item has a light gray background color and 10px padding.

**Conclusion:**

The key difference between Grid and Flexbox is that Grid creates a two-dimensional layout with rows and columns, while Flexbox creates a one-dimensional layout along a single axis. Grid is more suitable for complex grid-based designs, while Flexbox is more suitable for simpler, flexible layouts.

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

**example.**

Here's an detailed comparision of absolute and relative and sticky and fixed position along with example code to illustrate their differences:

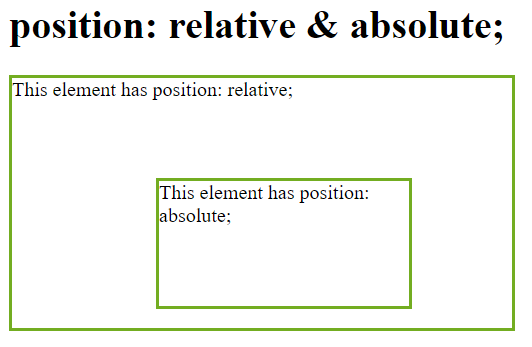
**Absolute Position:**

* When an element is positioned absolutely, it is removed from the normal document flow and positioned relative to its closest positioned ancestor.
* If there is no positioned ancestor, it is positioned relative to the initial containing block, usually the viewport.
* Absolute positioning allows precise control over the element's placement, as its position can be specified using the top, right, bottom, and left properties.

**Relative Position:**

* When an element is positioned relatively, it remains in the normal document flow, but its position can be adjusted relative to its normal position.
* The element's position can be changed using the top, right, bottom, and left properties, which move the element from its original position.

Example of Relative and Absolute positioning:



<!DOCTYPE html>

<html>

<head>

<style>

div.relative {

position: relative;

width: 400px;

height: 200px;

border: 3px solid #73AD21;

}

div.absolute {

position: absolute;

top: 80px;

right: 80px;

width: 200px;

height: 100px;

border: 3px solid #73AD21;

}

</style>

</head>

<body>

<h1>position: absolute;</h1>

<div class="relative">

This element has position: relative;

<div class="absolute">

This element has position: absolute;

</div>

</div>

</body>

</html>

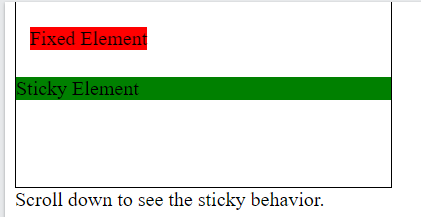
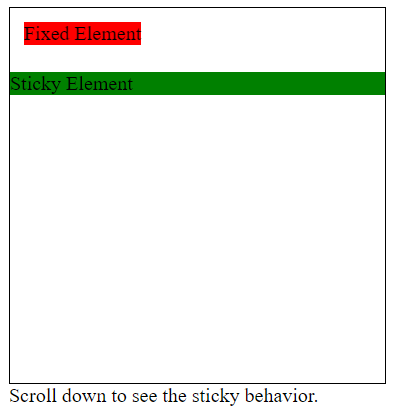
**Sticky Position:**

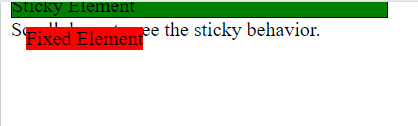
* Sticky positioning is a hybrid of relative and fixed positioning.
* The element is positioned according to the normal flow until it reaches a specified threshold, after which it becomes "stuck" and remains fixed in that position.
* The element switches between relative and fixed positions based on the user's scroll position.

**Fixed Position:**

* When an element is positioned fixed, it is removed from the normal document flow and remains fixed at a specified position relative to the viewport.
* The element's position is specified using the top, right, bottom, and left properties.

Example of Sticky and Fixed positioning:





<!DOCTYPE html>

<html>

<head>

<style>

  .container {

    height: 300px;

    width: 300px;

    border: 1px solid black;

  }

  .fixed-element {

    position: fixed;

    top: 20px;

    left: 20px;

    background-color: red;

  }

  .sticky-element {

    position: sticky;

    top: 60px;

    background-color: green;

  }

  .content {

    height: 1000px;

  }

</style>

</head>

<body>

<div class="container">

  <div class="fixed-element">Fixed Element</div>

  <div class="sticky-element">Sticky Element</div>

</div>

<div class="content">

  Scroll down to see the sticky behavior.

</div>

</body>

</html>

**Static Positioning:**

* Elements with position: static (which is the default) are positioned according to the normal flow of the document.
* Static positioned elements are not affected by the top, right, bottom, or left properties, as they do not have an offset from their default position.
* Static elements are not removed from the normal document flow, and they will appear in their default position as defined by the HTML structure.