Q.1 What’s Box Model in CSS ?

The box model in CSS is a fundamental concept that describes how elements are rendered and displayed on a web page. It defines the structure and layout of an element by considering it as a rectangular box with four main components: content, padding, border, and margin.

Here's a breakdown of each component:

Content: It represents the actual content of the element, such as text, images, or other HTML elements. The content area's size is determined by the element's width and height properties.

Padding: Padding is the space between the content and the element's border. It provides a cushioning effect by creating an inner space around the content. The padding's size is controlled using the padding property.

Border: The border surrounds the content and padding, and it defines the visual boundary of an element. It can have a specific thickness, style, and color, which are set using the border property.

Margin: The margin is the space outside the border that separates an element from other elements on the page. It creates an invisible gap around the element. The margin's size is set using the margin property.

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 specific elements or groups of elements on a web page. Each type of selector has its own advantages and use cases. Here are some commonly used selectors:

Element selectors: These are the most basic type of selectors and target elements based on their HTML tag name. For example, to select all <h1> headings, you would use the selector h1. The advantage of element selectors is that they are simple and can be applied to multiple elements of the same type throughout the page.

Class selectors: Class selectors target elements based on their HTML class attribute. You assign a specific class to an element using the class attribute and then use a period (.) followed by the class name in your CSS to target those elements. For example, .my-class will select all elements with the class "my-class". The advantage of class selectors is that they allow you to target specific groups of elements, regardless of their tag name. You can apply the same class to multiple elements and style them consistently.

ID selectors: ID selectors target elements based on their HTML id attribute. An id must be unique within the entire HTML document. To select an element by its id, you use a hash (#) followed by the id name. For example, #my-id will select the element with the id "my-id". The advantage of id selectors is that they provide a highly specific way to target individual elements. However, you should use them sparingly because an id can only be assigned to one element.

Attribute selectors: Attribute selectors target elements based on their HTML attribute values. They allow you to select elements with specific attribute values or patterns. For example, [type="text"] will select all elements with the attribute type set to "text". The advantage of attribute selectors is that they provide flexibility in targeting elements based on specific attributes or attribute values.

Pseudo-classes and pseudo-elements: Pseudo-classes and pseudo-elements are used to target elements based on a specific state or position within the document. Pseudo-classes are prefixed with a colon (:) and target elements based on events or conditions. Examples include :hover to select elements when the mouse is over them and :nth-child(n) to select elements based on their position in a parent container. Pseudo-elements, on the other hand, are used to style a specific part of an element. Examples include ::before and ::after to add content before or after an element.

The advantage of pseudo-classes and pseudo-elements is that they provide additional control and styling options for specific states or parts of elements.

Q.3 What is VW/VH ?

VW and VH are units of measurement in CSS that represent a percentage of the viewport width and viewport height, respectively. The viewport refers to the visible area of a web page in the browser window.

Here's a brief explanation of VW and VH:

VW (Viewport Width): 1 VW is equal to 1% of the width of the viewport. For example, if the viewport width is 1000 pixels, 1 VW would be equal to 10 pixels (1% of 1000 pixels). VW units allow you to set CSS properties relative to the width of the viewport.

VH (Viewport Height): 1 VH is equal to 1% of the height of the viewport. For example, if the viewport height is 800 pixels, 1 VH would be equal to 8 pixels (1% of 800 pixels). VH units allow you to set CSS properties relative to the height of the viewport.

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

|  | **Inline** | **Inline-Block** | **Block** |
| --- | --- | --- | --- |
| Display | Elements are displayed in a line | Elements are displayed in a line | Elements start on a new line |
| Width | Width is determined by content | Width can be set and respects other elements' horizontal space | Width expands to fill available horizontal space |
| Height | Height is determined by content | Height can be set, but vertical spacing may not apply | Height expands to fit content or as specified |
| Margin | Horizontal margin applies | Horizontal margin applies | Horizontal margin applies |
|  |  |  |  |

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

|  |  |  |
| --- | --- | --- |
|  | Content-box | Border-box |
| Calculation | Width and height properties exclude padding and border | Width and height properties include padding and border |
| Total Size | Content size + Padding size + Border size | Specified width and height include padding and border |
| Adjustment | Padding and border increase the overall size of the element | Padding and border are part of the specified width and height |
| Simplified Example | **width: 200px;** with **padding: 20px; border: 2px solid;** | **width: 200px;** includes padding and border in the 200px size |

Q6. What’s z-index and How does it Function ?

The z-index property in CSS controls the stacking order of positioned elements on a web page. It determines the vertical placement of elements along the z-axis, where higher values indicate a higher stacking order. Elements with a higher z-index value will appear in front of elements with a lower value within the same stacking context.

Key points about z-index:

Elements with a higher z-index value are positioned in front of elements with a lower value.

Sibling elements are compared first, and the one with the higher z-index value appears on top.

Stacking contexts are created by positioned elements, and elements within the same stacking context are ordered based on their z-index values.

Stacking order can be influenced by parent-child relationships and the hierarchy of stacking contexts.

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

Grid:

CSS Grid is a two-dimensional layout system that allows you to create grid-like structures of rows and columns. It provides powerful control over layout, enabling complex multi-dimensional designs. Grid layouts consist of a container and its child items, with properties to define the size and behavior of tracks (rows or columns) and position cells within the grid.

Flexbox:

Flexbox is a one-dimensional layout system that arranges elements in a row or column. It offers efficient space distribution among items within a container. Flexbox operates along a main axis and cross axis, allowing you to control alignment, sizing, and order of elements. It is commonly used for flexible and responsive layouts.

|  | **Flexbox** | **CSS Grid** |
| --- | --- | --- |
| Dimensionality | One-dimensional layout system | Two-dimensional layout system |
| Layout Control | Simpler, suitable for single row/column | More advanced control over layout |
| Alignment | Powerful alignment capabilities | Fine-grained control over alignment |
| Nesting Support | No direct support for nested layouts | Support for nested grids |
| Browser Support | Widely supported across modern browsers | Widely supported across modern browsers |
| Use Cases | Ideal for linear layouts | Ideal for grid-like structures |
| Learning Curve | Relatively easier to learn and understand | Requires understanding of grid concepts |
|  |  |  |

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

Absolute Positioning:

Absolute positioning allows you to precisely position an element relative to its closest positioned ancestor or the document itself.

Example:

.parent {

position: relative;

height: 200px;

width: 200px;

}

.child {

position: absolute;

top: 50px;

left: 50px;

}

Relative Positioning:

Relative positioning allows you to position an element relative to its normal position in the document flow.

Example:

css

Copy code

.element {

position: relative;

top: 20px;

left: 30px;

}

In this example, the element is positioned 20 pixels down and 30 pixels to the right from its original position within the document flow.

Sticky Positioning:

Sticky positioning is a hybrid of relative and fixed positioning. The element behaves as relatively positioned until it reaches a specific threshold, then it becomes fixed in that position.

Example:

css

Copy code

.element {

position: sticky;

top: 50px;

}

In this example, the element will behave as relatively positioned until the top of the viewport reaches 50 pixels, at which point it becomes fixed in that position.

Fixed Positioning:

Fixed positioning allows you to position an element relative to the browser window, regardless of scrolling.

Example:

.element {

position: fixed;

top: 10px;

right: 10px;

}

In this example, the element is positioned 10 pixels from the top and right of the browser window.

These positioning properties provide different ways to control the placement and behavior of elements on a web page. Absolute and relative positioning offer relative placement within the document flow, while sticky and fixed positioning allow for fixed placement in relation to the viewport.

Relative Positioning:

Relative positioning allows you to position an element relative to its normal position in the document flow.

Example:

.element {

position: relative;

top: 20px;

left: 30px;

}

Sticky Positioning:

Sticky positioning is a hybrid of relative and fixed positioning. The element behaves as relatively positioned until it reaches a specific threshold, then it becomes fixed in that position.

Example:

.element {

position: sticky;

top: 50px;

}

Q.9 Build Periodic Table as shown in the below image

Ans: please copy below link and paste url on your browser to see output

<https://codepen.io/rup08/pen/LYXzQyN>

**Q.10** Build Responsive Layout both desktop and mobile and Tablet, see below image for reference ?

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<style>

\* {

box-sizing: border-box;

}

.menu {

float: left;

width: 20%;

text-align: center;

}

.menu a {

background-color: #e5e5e5;

padding: 8px;

margin-top: 7px;

display: block;

width: 100%;

color: black;

}

.main {

float: left;

width: 60%;

padding: 0 20px;

}

.right {

background-color: #e5e5e5;

float: left;

width: 20%;

padding: 15px;

margin-top: 7px;

text-align: center;

}

@media only screen and (max-width: 620px) {

/\* For mobile phones: \*/

.menu, .main, .right {

width: 100%;

}

}

</style>

</head>

<body style="font-family:Verdana;color:#aaaaaa;">

<div style="background-color:#e5e5e5;padding:15px;text-align:center;">

<h1>Hello World</h1>

</div>

<div style="overflow:auto">

<div class="menu">

<a href="#">Link 1</a>

<a href="#">Link 2</a>

<a href="#">Link 3</a>

<a href="#">Link 4</a>

</div>

<div class="main">

<h2>Lorum Ipsum</h2>

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>

</div>

<div class="right">

<h2>About</h2>

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>

</div>

</div>

</body>

</html>