Q.1 What's Box Model in CSS?

The Box Model is a fundamental concept in CSS (Cascading Style Sheets) that defines how elements on a web page are rendered and displayed. It describes the structure of an element by dividing it into several layers or boxes. These layers include the content area, padding, border, and margin.

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

CSS provides several types of selectors that allow you to target specific elements on a web page for styling. Here are some commonly used selectors:

Element Selector: Selects elements based on their HTML tag name. For example, to target all paragraphs, you can use the selector `p`.
Class Selector: Selects elements based on the value of their `class` attribute. Classes are reusable and can be applied to multiple elements. For example, to target elements with a class of "highlight," you can use the selector `.highlight`.
ID Selector: Selects a single element based on its unique `id` attribute. IDs should be unique within the HTML document. For example, to target an element with the ID "logo," you can use the selector `#logo`.
Attribute Selector: Selects elements based on the presence or value of their attributes. For example, to target all `input` elements with a `type` attribute of "checkbox," you can use the selector `input[type="checkbox"]`.
Descendant Selector: Selects elements that are descendants of another element. It matches elements that are inside a specific parent element. For example, to target all `span` elements inside a `div`, you can use the selector `div span`.
Pseudo-class Selector: Selects elements based on a specific state or condition. For example, the `:hover` pseudo-class selects elements when they are being hovered over by the mouse cursor.

The advantages of using different types of selectors include improved specificity, reusability, and the ability to target specific elements or groups of elements. Selectors allow you to apply styles to elements more precisely and create consistent styles throughout your web page or application.

Q.3 What is VW/VH?

VW (Viewport Width) and VH (Viewport Height) are units of measurement in CSS that allow you to specify sizes relative to the dimensions of the browser viewport.

- VW (Viewport Width): It represents 1% of the viewport's width. For example, if the viewport is 1000 pixels wide, 1vw would be equal to 10 pixels. VW units are useful for creating responsive layouts that scale proportionally based on the width of the viewport.
- VH (Viewport Height): It represents 1% of the viewport's height. For example, if the viewport is 800 pixels high, 1vh would be equal to 8 pixels. VH units are helpful for sizing elements based on the height of the viewport.

💡 Q.4 What's the difference between Inline, Inline Block, and Block?

> Inline: Inline elements are displayed within the normal flow of text, occupying only the space necessary for their content. They do not cause line breaks and cannot have explicit width and height. Examples of inline elements include `<span>`, `<a>`, and `<strong>`.
> Inline Block: Inline-block elements are similar to inline elements in that they flow within the text content, but they can have explicit width, height, padding, and margin. They also respect line breaks and can be positioned vertically using vertical-align. Examples of inline-block elements include `<img>` and `<input>`.
> Block: Block elements, by default, start on a new line and occupy the full width available. They can have explicit width, height, padding, and margin. Block elements create a block-level box. Examples of block-level elements include `<div>`, `<p>`, and `<h1>` to `<h6>`.

.

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

In CSS, the `box-sizing` property defines how the width and height of an element are calculated, including its padding and border. There are two possible values for `box-sizing`:

> `content-box` (default): The width and height of an element only include the content area and do not account for the padding or border. In this model, if you set a width of 300 pixels, for example, the element's total width will be 300 pixels plus any additional padding or border.

> `border-box`: The width and height of an element include the content area, padding, and border. In this model, if you set a width of 300 pixels, the element's total width will be 300 pixels, and the padding and border will be included within that width.

💡 Q.6 What's z-index and how does it function?

The `z-index` property in CSS is used to control the stacking order of positioned elements on a web page. It determines the visual hierarchy and layering of elements in the z-axis, which is the axis perpendicular to the screen.

The `z-index` property accepts an integer value, where a higher value indicates a higher stacking order. Elements with a higher `z-index` value will appear on top of elements with a lower value. If two elements have the same `z-index`, the order of appearance in the HTML document determines which element appears on top.

💡 Q.7 What's Grid & Flex, and what's the difference between them?

Both Grid and Flex are layout systems in CSS that allow you to create responsive and flexible designs, but they have different approaches and use cases.

> Grid (CSS Grid Layout): CSS Grid Layout is a two-dimensional layout system that divides the available space into rows and columns. It allows you to create complex grid structures and precisely control the placement and alignment of elements within the grid. Grid provides a powerful way to create both simple and complex layouts, such as magazine-style designs or multi-column forms. It works well for overall page layouts.
> Flexbox (CSS Flexible Box Layout): CSS Flexbox is a one-dimensional layout system that focuses on arranging elements in a single row or column. It provides an intuitive way to distribute space, align elements, and handle responsive layouts. Flexbox is especially useful for creating flexible and dynamic layouts, such as navigation menus, flexible content containers, or vertically aligning elements. It works well for individual components or smaller sections of a page.

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

Absolute Positioning: When an element is positioned absolutely, it is removed from the normal document flow and positioned relative to its closest positioned ancestor (or the initial containing block if no ancestor is positioned). Absolute positioning allows precise control over an element's placement. It is often used for overlays, tooltips, or elements that need to be positioned exactly in a specific location.

Relative Positioning: When an element is positioned relatively, it is still a part of the normal document flow but can be adjusted relative to its normal position. It can be moved using the `top`, `right`, `bottom`, and `left` properties. Relative positioning is useful for making slight adjustments to the position of an element without disrupting the surrounding content.

Sticky Positioning: Sticky positioning is a hybrid of relative and fixed positioning. An element with sticky positioning is initially positioned according to the normal flow of the document but becomes fixed (stays in place) as the user scrolls past a specified threshold. Sticky positioning is often used for navigation bars or elements that should stick to a specific position on the page.

Fixed Positioning: When an element is positioned fixed, it is removed from the normal document flow and stays in a fixed position relative to the browser viewport. Even if the page is scrolled, the fixed element remains in the same place. Fixed positioning is commonly used for elements that should be visible at all times, such as a header or a footer.

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

```
<!DOCTYPE html>
<html>
<head>
 <title>Periodic Table</title>
 <style>
  table {
   border-collapse: collapse;
  }
  td {
   border: 1px solid black;
   padding: 10px;
  }
  .group-1 {
   background-color: #ffcccc;
```

```
    }
    .group-2 {
      background-color: #ffcc99;
    }
    /* Add more styles for other groups/colors */
  </style>
</head>
<body>
  <table>
    <tr>
      <td class="group-1">H<br>1</td>
      <td class="group-1">Li<br>3</td>
      <td></td>
      <td></td>
      <td></td>
      <!-- Add more elements in the first row -->
    </tr>
    <tr>
      <td class="group-2">Be<br>4</td>
      <td class="group-1">B<br>5</td>
      <td class="group-1">C<br>6</td>
      <td class="group-1">N<br>7</td>
      <td class="group-1">O<br>8</td>
      <!-- Add more elements in the second row -->
    </tr>
    <!-- Add more rows for other periods -->
  </table>
</body>
</html>
```

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

```
<!DOCTYPE html>
<html>
<head>
  <title>Responsive Layout Example</title>
```

```
<style>
  /* Base styles for the layout */
  body {
    margin: 0;
    padding: 0;
    font-family: Arial, sans-serif;
  }
  .container {
    max-width: 1200px;
    margin: 0 auto;
    padding: 20px;
  }
  .box {
    background-color: lightgray;
    padding: 20px;
    margin-bottom: 20px;
  }

  /* Media queries for different screen sizes */
  @media (max-width: 767px) {
    /* Mobile styles */
    .box {
      background-color: lightblue;
    }
  }

  @media (min-width: 768px) and (max-width: 1023px) {
    /* Tablet styles */
    .box {
      background-color: lightgreen;
    }
  }

  @media (min-width: 1024px) {
    /* Desktop styles */
    .box {
      background-color: lightyellow;
    }
  }
```

```
  </style>
</head>
<body>
  <div class="container">
    <div class="box">
      <h2>Box 1</h2>
      <p>This is the first box in the layout.</p>
    </div>
    <div class="box">
      <h2>Box 2</h2>
      <p>This is the second box in the layout.</p>
    </div>
    <div class="box">
      <h2>Box 3</h2>
      <p>This is the third box in the layout.</p>
    </div>
  </div>
</body>
</html>
```