Assignment Questions 2

01	**What's Box Model in CSS ?** The box model is a fundamental concept in CSS (Cascading Style Sheets) that describes how elements are displayed and structured on a webpage. It consists of four main components: 1. Content: It refers to the actual content of an element, such as text, images, or other media. 2. Padding: Padding is the space between the content and the border of an element. It provides an internal spacing within the element, adding whitespace between the content and the border. 3. Border: The border surrounds the padding and content of an element and can have a specific style, width, and color. It creates a visual boundary between the content and the margin. 4. Margin: Margin is the space outside the border of an element. It provides spacing between elements and determines the distance between an element and its neighboring elements. Various CSS properties are associated with the box model and allow you to control its different components. Some of the key properties include: 1. width: Specifies the width of the content area. 2. height: Sets the height of the content area. 3. padding: Defines the spacing between the content and the border. 4. border: Determines the style, width, and color of the border. 5. margin: Controls the spacing between the element and its neighboring elements. 6. box-sizing: Specifies how the total width and height of an element are calculated, including or excluding the padding and border. These are just a few examples, and there are many more properties related to the box model that allow you to fine-tune the layout and spacing of elements on a webpage.
02	**What are the Different Types of Selectors in CSS & what are the advantages of them?** CSS selectors are used to target specific elements on a webpage and apply styles to them. There are several types of selectors available in CSS, each with its own syntax and functionality. Here are some of the commonly used types of selectors: **1**. **Element Selectors**: They target elements based on their HTML tag name. For example, using `p` as a selector will target all `<p>` elements on the page. **Advantages**: Element selectors are simple and easy to use. They allow you to apply styles to all

	elements of a specific type throughout your document. **2. Class Selectors:** They target elements that have a specific class attribute assigned to them. Class selectors are denoted by a period (.) followed by the class name. For example, `.highlight` will target all elements with the class "highlight" assigned to them. **Advantages**: Class selectors offer more flexibility as they can be applied to multiple elements throughout a page or even across multiple pages. They allow you to group elements and apply consistent styles to them. **3. ID Selectors:** They target a single element based on its unique ID attribute. ID selectors are denoted by a hash (#) followed by the ID name. For example, `#logo` will target the element with the ID "logo". **Advantages:** ID selectors are highly specific and useful when you want to apply styles to a unique element. They have higher specificity than other selectors, making them useful for overriding styles applied by other selectors. **4. Attribute Selectors:** They target elements based on their attribute values. Attribute selectors can be used to select elements with specific attributes, attribute values, or attribute value prefixes/suffixes. For example, `[type="submit"]` will target all elements with the attribute `type` set to "submit". **Advantages**: Attribute selectors allow you to target elements based on specific attribute values, making them useful when you want to style elements with certain characteristics. **5. Pseudo-classes and Pseudo-elements:** Pseudo-classes and pseudo-elements target elements based on their state or position in the document structure. For example, `:hover` targets an element when the mouse is hovering over it, and `::before` creates a pseudo-element before the content of an element. **Advantages**: Pseudo-classes and pseudo-elements provide additional control and styling options. They allow you to apply styles to elements based on user interactions or create dynamic effects.
03	**What is VW/VH ?** VW (viewport width) and VH (viewport height) are units of measurement in CSS that are relative to the size of the viewport, which is the visible area of a web page. **1. VW (Viewport Width):** The VW unit represents a percentage of the viewport's width. For example, 1vw is equal to 1% of the viewport width. If the viewport is 1000 pixels wide, 1vw would be equal to 10 pixels. **2. VH (Viewport Height):** The VH unit represents a percentage of the viewport's height. Similarly, 1vh is equal to 1% of the viewport height. If the viewport is 800 pixels tall, 1vh would be equal to 8 pixels. On the other hand, PX (pixels) is an absolute unit of measurement that represents a fixed

<table>
<tr><td></td><td>number of pixels.

The main difference between VW/VH and PX is that VW/VH are relative units based on the size of the viewport, while PX is an absolute unit that represents a fixed pixel value.

In contrast, PX units are fixed and do not adjust based on the viewport size. They provide more precise control over element sizes, but they may not respond well to different screen sizes or device orientations.</td></tr>
<tr><td>04</td><td>Whats difference between Inline, Inline Block and block ?

In CSS, "inline," "inline-block," and "block" are three display properties that determine how elements are rendered and how they interact with other elements on a webpage.

1. Inline:
- Inline elements are displayed inline within a block-level element or other inline elements.
- They do not start on a new line and only take up the necessary width and height to contain their content.
- Examples of inline elements include `<span>`, `<a>`, `<em>`, `<strong>`, and `<img>`.
- Inline elements do not have width and height properties. You cannot set a fixed width or height on an inline element.

2. Inline-block:
- Inline-block elements are displayed inline like inline elements, but they also retain block-level properties, such as being able to set width, height, padding, and margins.
- They start on the same line as other inline elements but can have their own width and height properties, allowing them to be sized and positioned like block-level elements.
- Inline-block is often used for creating layouts or aligning elements horizontally while still retaining the ability to set dimensions and apply padding and margins.
- Examples of inline-block elements include `<input>`, `<button>`, and `<div>` when its display property is set to inline-block.

3. Block:
- Block elements are displayed as individual blocks that start on a new line and take up the entire width of their parent container by default.
- They create a line break before and after themselves, causing subsequent elements to appear on new lines.
- Block elements have their own width and height properties, and you can set margins, padding, and borders to control their spacing and appearance.
- Examples of block elements include `<div>`, `<p>`, `<h1>` to `<h6>`, `<ul>`, `<li>`, and `<section>`.</td></tr>
<tr><td>05</td><td>How is Border-box different from Content Box?

<table>
<tr><th>Property</th><th>Content Box</th><th>Border Box</th></tr>
<tr><td>Calculation</td><td>Width and height are calculated based on the</td><td>Width and height are calculated included padding</td></tr>
</table>
</td></tr>
</table>

	content size.	and border sizes
Box Model	Follows the standard CSS box model	Follows an alternative box model
Total width	Content width + padding + border + margin	Width includes only the content size
Total Height	Content height + padding + border + margin	Height includes only the content size
Padding and Border	Added to the specified width and height	Included in the specified width and height
Usage	Commonly used for standard CSS layouts and elements	Often used for more efficient and predictable sizing

06

What's z-index and How does it Function ?

The "z-index" property in CSS controls the stacking order of positioned elements along the z-axis, determining which elements appear in front of or behind other elements. It works in conjunction with the positioning (e.g., relative, absolute, fixed) of elements.

"z-index" property functions:

1. Stacking Context: Each positioned element (position value other than "static") with a specified "z-index" value creates a stacking context. The stacking context establishes a local coordinate system for its descendants, affecting how elements are stacked within that context.

2. Higher Values: Elements with higher "z-index" values are positioned in front of elements with lower values or no specified "z-index" value. If two elements have the same "z-index," the stacking order follows the order of their placement in the HTML markup.

3. Parent-Child Relationship: A child element within a stacking context cannot appear in front of its parent, even if the child has a higher "z-index" value. However, the child can be positioned in front of other siblings within the same stacking context if their "z-index" values permit it.

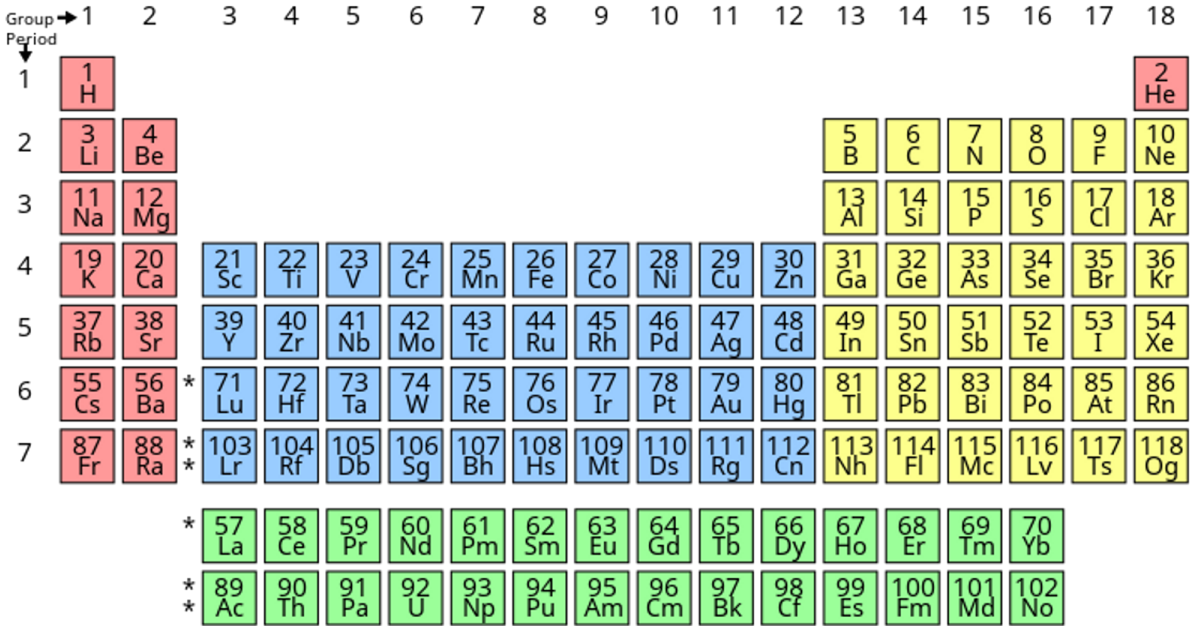
4. Stacking Order: Stacking order is determined not only by the "z-index" but also by the document tree structure. Elements that are later in the document tree (HTML markup) are stacked in front of earlier elements unless overridden by "z-index" values.

5. Negative Values: Negative "z-index" values are allowed and can position elements behind the default stacking level. Elements with negative "z-index" values are stacked behind elements with positive or zero values.

It's important to note that "z-index" only applies to positioned elements. Elements with a "static" position value will ignore the "z-index" property.

	By manipulating the "z-index" property, you can control the layering and visual hierarchy of elements, allowing you to position certain elements in front or behind others to achieve desired visual effects or stacking arrangements on a webpage.
07	**What's Grid & Flex and difference between them?** Grid and Flex are both CSS layout systems used for creating responsive and flexible designs, but they have different approaches and use cases: **1. Grid Layout:** - Grid layout is a two-dimensional system that allows you to create complex grid structures with rows and columns. - It enables precise control over the placement and alignment of elements within a grid container. - Grid layout is best suited for designing complex layouts that require more control over both the horizontal and vertical axes. - It provides features such as defining fixed or flexible column and row sizes, creating grid tracks, and positioning items within specific grid cells. - Grid layout is useful for creating grid-based designs, such as magazine-style layouts, card-based interfaces, or overall page structures. **2. Flexbox Layout:** - Flexbox layout is a one-dimensional system that focuses on arranging elements along a single axis (either horizontally or vertically). - It provides a flexible way to distribute space among elements within a flex container, making it easier to create dynamic and responsive layouts. - Flexbox is best suited for simpler layouts or arranging items within a container in a single row or column. - It offers features such as defining flexible item sizes, controlling the order and alignment of items, and distributing space and alignment within the container. - Flexbox layout is commonly used for creating navigation menus, form layouts, card grids, or centering elements within a container. In summary, the main differences between Grid and Flexbox are: - Grid is a two-dimensional layout system, while Flexbox is a one-dimensional layout system. - Grid allows for complex grid structures with rows and columns, while Flexbox focuses on arranging items along a single axis. - Grid provides precise control over both horizontal and vertical alignment, while Flexbox focuses on alignment within a single axis. - Grid is suitable for complex layouts requiring control over both axes, while Flexbox is ideal for simpler layouts or aligning items within a single axis. Depending on your layout requirements, you can choose between Grid and Flexbox or even combine them to create more versatile and powerful layouts.
08	Difference between absolute and relative and sticky and fixed position explain with example. **Positioning** | **Description** | **Behaviour** | **Example**

Absolute	Positioned relative to the nearest positioned ancestor	Takes the element out of the normal flow of the document. - Positioned relative to its closest positioned ancestor or the initial containing block if no ancestor is positioned. - Does not leave a space in the document flow where it would have been positioned.	inside a <section> with "position: relative"
Relative	Positioned relative to its normal position in the document	Does not take the element out of the normal flow of the document. - Positioned relative to its normal position. - Surrounding elements are not affected by its positioning.	<img> with "position: relative"
Sticky	Acts like ‘relative’ until it reaches a specified threshold	Initially behaves like "relative" positioning, but becomes "fixed" when the specified threshold (e.g., scrolling) is reached. - Stays in the specified position relative to the viewport. - Can create a sticky element that remains visible while scrolling within a container.	<nav> with "position: sticky" and "top: 0"
Fixed	Positioned relative to the viewport	Does not scroll with the page and remains in the specified position relative to the viewport. - Does not affect the layout	with "position: fixed" and "top: 20px"

	<table><tr><td></td><td></td><td>of surrounding elements.</td><td></td></tr></table>
09	**Build Periodic Table as shown in the below image**  Github Link → [Period Table Link]
10	Build Responsive Layout both desktop and mobile and Tablet, see below image for reference ?

	 Github link → [Responsive layout link]