**Assignment 1(6 July, 2023)**

**Q.1 What’s Box Model in CSS ?**

Ans. All HTML elements can be considered as boxes.

The CSS Box Model

In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:

Explanation of the different parts:

* **Content** - The content of the box, where text and images appear
* **Padding** - Clears an area around the content. The padding is transparent
* **Border** - A border that goes around the padding and content
* **Margin** - Clears an area outside the border. The margin is transparent

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

**Ans.** CSS selectors are used to "find" (or select) the HTML elements you want to style.

We can divide CSS selectors into five categories:

* Simple selectors (select elements based on name, id, class)
* [Combinator selectors](https://www.w3schools.com/CSS/css_combinators.asp) (select elements based on a specific relationship between them)
* [Pseudo-class selectors](https://www.w3schools.com/CSS/css_pseudo_classes.asp) (select elements based on a certain state)
* [Pseudo-elements selectors](https://www.w3schools.com/CSS/css_pseudo_elements.asp) (select and style a part of an element)
* [Attribute selectors](https://www.w3schools.com/CSS/css_attribute_selectors.asp) (select elements based on an attribute or attribute value)

1. **The CSS element Selector :-** The element selector selects HTML elements based on the element name.

Example

Here, all <p> elements on the page will be center-aligned, with a red text color:

p {  
  text-align: center;  
  color: red;  
}

1. **The CSS id Selector :-**

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

Example

The CSS rule below will be applied to the HTML element with id="para1":

#para1 {  
  text-align: center;  
  color: red;  
}

1. **The CSS class Selector**

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name.

**Example:**

In this example all HTML elements with class="center" will be red and center-aligned:

.center {  
  text-align: center;  
  color: red;  
}

You can also specify that only specific HTML elements should be affected by a class.

**Example :**

In this example only <p> elements with class="center" will be red and center-aligned:

p.center {  
  text-align: center;  
  color: red;  
}

HTML elements can also refer to more than one class.

**Example :-**

In this example the <p> element will be styled according to class="center" and to class="large":

<p class="center large">This paragraph refers to two classes.</p>

1. **The CSS Universal Selector**

The universal selector (\*) selects all HTML elements on the page.

**Example :-**

The CSS rule below will affect every HTML element on the page:

\* {  
  text-align: center;  
  color: blue;  
}

The CSS Grouping Selector

The grouping selector selects all the HTML elements with the same style definitions.

Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

h1 {  
  text-align: center;  
  color: red;  
}  
  
h2 {  
  text-align: center;  
  color: red;  
}  
  
p {  
  text-align: center;  
  color: red;  
}

It will be better to group the selectors, to minimize the code.

To group selectors, separate each selector with a comma.

**Example:**

In this example we have grouped the selectors from the code above:

h1, h2, p {  
  text-align: center;  
  color: red;  
}

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

**Ans.** CSS Units CSS has several different units for expressing a length.

Many CSS properties take "length" values, such as width, margin, padding, font-size, etc.

**Length** is a number followed by a length unit, such as 10px, 2em, etc.

vh and vw are CSS units that represent the height and width of the viewport.

Viewport Height (vh): This unit is based on the height of the viewport. A value of 1vh is equal to 1% of the viewport height.

Viewport Width (vw): This unit is based on the width of the viewport. A value of 1vw is equal to 1% of the viewport width.

vmin and vmax: These units are related to the maximum or minimum value of vh and vw.

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

**Ans. Inline elements:**

1. respect left & right margins and padding, but **not** top & bottom
2. **cannot** have a width and height set
3. allow other elements to sit to their left and right.
4. see very important side notes on this [here](https://hacks.mozilla.org/2015/03/understanding-inline-box-model/).

**Block elements:**

1. respect all of those
2. force a line break after the block element
3. acquires full-width if width not defined

**Inline-block elements:**

1. allow other elements to sit to their left and right
2. respect top & bottom margins and padding
3. respect height and width

* An inline element has no line break before or after it, and it tolerates HTML elements next to it.
* A block element has some whitespace above and below it and does not tolerate any HTML elements next to it.
* An inline-block element is placed as an inline element (on the same line as adjacent content), but it behaves as a block element.

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

**Ans.** ***border-box***and ***content-box*** are the two different values of [***box-sizing***](https://www.geeksforgeeks.org/css-box-sizing-property/).

* **content-box:**This is the default value of box-sizing. The dimension of element only includes ‘height’ and ‘width’ and does not include ‘border’ and ‘padding’ given to element. Padding and Border take space outside the element.

Example:

<html>

<head>

<style>

.box {

width: 300px;

height: 200px;

padding: 15px;

border: 10px solid black;

box-sizing: content-box;

background: red;

display: inline-block;

}

</style>

</head>

<body>

<h1 class="box">PW SKILLS</h1>

</body>

</html>

**Note** :- border-box = size of content + padding + border

In the above scenario,

width = width of content

border-box width = width of content + left padding + right padding + left border + right border

**Syntax:**

box-sizing: content-box;

* **border-box:**In this value, not only width and height properties are included but you will find padding and border inside of the box for example .box {width: 200px; border: 10px solid black;} renders a box that is 200px wide.

**Example:**

<html>

<head>

<style>

.box {

width: 200px;

height: 200px;

padding: 15px;

border: 10px solid green;

box-sizing: border-box;

background: red;

display: inline-block;

}

</style>

</head>

<body>

<h1 class="box">Heading 1</h1>

</body>

</html>

Calculation of dimension

width = border + padding + width of the content,

height = border + padding + height of the content.

**Syntax:**

box-sizing: border-box;

**Note :-** *when using****box-sizing : content-box;****the content size remain same while border-box size grows as padding and border grow. but when using****box-sizing: border-box;****, the size of border-box remains same while size of content decreases as padding and border grow.*

**Example:**In the below example, we will see the difference between the width when ***content-box*** and ***border-box*** properties are applied.

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

**Ans.** The z-index property specifies the stack order of an element.

An element with greater stack order is always in front of an element with a lower stack order.

Note: z-index only works on positioned elements (position: absolute, position: relative, position: fixed, or position: sticky) and flex items (elements that are direct children of display:flex elements).

Note: If two positioned elements overlap without a z-index specified, the element positioned last in the HTML code will be shown on top.

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

**Ans.**

[**Grid**](https://www.geeksforgeeks.org/css-grid-property/)**:** CSS Grid Layout, is a two-dimensional grid-based layout system with rows and columns, making it easier to design web pages without having to use floats and positioning. Like tables, grid layout allow us to align elements into columns and rows.

To get started you have to define a container element as a grid with **display: grid,** set the column and row sizes with grid-template-columns and grid-template-rows, and then place its child elements into the grid with grid-column and grid-row.

[**Flexbox**](https://www.geeksforgeeks.org/introduction-to-css-flexbox/)**:** The CSS Flexbox offers a one-dimensional layout. It is helpful in allocating and aligning the space among items in a container (made of grids). It works with all kinds of display devices and screen sizes.

To get started you have to define a container element as a grid with **display: flex;**

**Difference Between Grid and Flexbox:**

**1. Dimensionality and Flexibility:**

* Flexbox offers greater control over alignment and space distribution between items. Being one-dimensional, Flexbox only deals with either columns or rows.
* Grid has two-dimension layout capabilities which allow flexible widths as a unit of length. This compensates for the limitations in Flex.

**2. Alignment:**

* Flex Direction allows developers to align elements vertically or horizontally, which is used when developers create and reverse rows or columns.
* CSS Grid deploys fractional measure units for grid fluidity and auto-keyword functionality to automatically adjust columns or rows.

**3. Item Management**

* Flex Container is the parent element while Flex Item represents the children. The Flex Container can ensure balanced representation by adjusting item dimensions. This allows developers to design for fluctuating screen sizes.
* Grid supports both implicit and explicit content placement. Its inbuilt automation allows it to automatically extend line items and copy values into the new creation from the preceding item.

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

Ans. **Difference between position: fixed and position: sticky property:**

| **S.No.** | **Position: fixed** | **Position: Sticky** |
| --- | --- | --- |
| 1. | Element with *position: fixed* property is fixed to the viewport and doesn’t move irrespective of scrolling. | Element with *position: sticky* property can scroll to an offset value provided by the user. |
| 2. | Element with position: fixed property never leaves the viewport position it was fixed to. | Element with position: sticky property leaves the viewport when its parent element scrolls off the viewport. |
| 3. | This property is supported by all the browsers. | This property is only supported by all modern browsers. |
| 4. | Element with position: fixed property does not effect the other element’s flow on the page ie it does not capture additional space. | Element with position: sticky property does effect the other element’s flow in the page ie., it will take the additional space. |

**Example: Fixed Position**

<!DOCTYPE html>

<html>

<head>

<style>

div.fixed {

position: fixed;

bottom: 0;

right: 0;

width: 300px;

border: 3px solid #73AD21;

}

</style>

</head>

<body>

<h2>position: fixed;</h2>

<p>An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled:</p>

<div class="fixed">

This div element has position: fixed;

</div>

</body>

</html>

**Example: Sticky Position**

An element with position: sticky; is positioned based on the user's scroll position.

A sticky element toggles between relative and fixed, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

<!DOCTYPE html>

<html>

<head>

<style>

div.sticky {

position: -webkit-sticky;

position: sticky;

top: 0;

padding: 5px;

background-color: #cae8ca;

border: 2px solid #4CAF50;

}

</style>

</head>

<body>

<p>Try to <b>scroll</b> inside this frame to understand how sticky positioning works.</p>

<div class="sticky">I am sticky!</div>

<div style="padding-bottom:2000px">

<p>In this example, the sticky element sticks to the top of the page (top: 0), when you reach its scroll position.</p>

<p>Scroll back up to remove the stickyness.</p>

<p>Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.</p>

<p>Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.</p>

</div>

</body>

</html>

**Difference between absolute and relative position:**

**Absolute position:-** This tells the browser that whatever is going to be positioned should be removed from the normal flow of the document and will be placed in an exact location on the page. It won't affect how the elements before it or after it in the HTML are positioned on the Web page however it *will* be subject to it's parents' positioning unless you override it.

Example:

<!DOCTYPE html>

<html>

<head>

<style>

div.relative {

position: relative;

width: 400px;

height: 200px;

border: 3px solid #73AD21;

}

div.absolute {

position: absolute;

top: 80px;

right: 0;

width: 200px;

height: 100px;

border: 3px solid #73AD21;

}

</style>

</head>

<body>

<h2>position: absolute;</h2>

<p>An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed):</p>

<div class="relative">This div element has position: relative;

<div class="absolute">This div element has position: absolute;</div>

</div>

</body>

</html>

**Relative Position:-** Relative positioning uses the same four positioning properties as absolute positioning. But instead of basing the position of the element upon the browser view port, it starts from where the element would be if it were still in the normal *flow*.

For example, if you have three paragraphs on your Web page, and the third has a position: relative style placed on it, its position will be offset based on its current location-- not from the original sides of the view port.

Paragraph 1.

Paragraph 2.

Paragraph 3.

In the above example, the third paragraph will be positioned 3em from the left side of the container element, but will still be below the first two paragraphs. It would remain in the normal flow of the document, and just be offset slightly. If you changed it to position: absolute;, anything following it would display on top of it, because it would no longer be in the normal flow of the document.