**HTML/ CSS related Questions**

**Q1:** Is border effect the dimensions of the element?

**Answer:** Yes, the **border** property in CSS can affect the dimensions of elements. The **border** property includes the width, style, and color of an element's border. When you apply a border to an element, it can increase the overall dimensions of the element.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<style>

.box {

width: 100px;

height: 100px;

border: 10px solid black;

}

</style>

<title>Border and Dimensions</title>

</head>

<body>

<div class="box"></div>

</body>

</html>

In this example:

* The **.box** class represents a square element with a width and height of 100 pixels.
* The **border** property is set to **10px solid black**, adding a black border around the element.

The total dimensions of the **.box** element, including the border, will be 120 pixels (100 pixels for the content + 10 pixels for the left border + 10 pixels for the right border).

**Q2:** As we know that float = center property does not exist then how we can place an element between two elements that are float=right and float=left?

**Answer:**

If you have two elements floating left and right, and you want to place another element between them, you can use a combination of techniques to achieve the desired layout. One common approach is to use the **clear** property to ensure that the center element is positioned below the floated elements.

Here's an example:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<style>

.left {

float: left;

width: 30%;

background-color: lightblue;

}

.right {

float: right;

width: 30%;

background-color: lightcoral;

}

.center {

margin: 0 auto; /\* Center the element \*/

width: 30%;

background-color: lightgreen;

}

/\* Clear the floats to ensure proper layout \*/

.clearfix::after {

content: "";

display: table;

clear: both;

}

</style>

<title>Floating Elements</title>

</head>

<body>

<div class="left">Left</div>

<div class="right">Right</div>

<!-- Use the clearfix class to clear floats -->

<div class="clearfix"></div>

<div class="center">Center</div>

</body>

</html>

* The **.left** and **.right** elements are floated left and right, respectively.
* The **.center** element is centered using **margin: 0 auto;**.
* The **.clearfix** class is added to clear the floats. This ensures that the center element is placed below the floated elements.

Using a clearfix is a common technique to handle floats and prevent layout issues. The **::after** pseudo-element creates a clearing element after the floated elements.

Remember that the specific widths and styling can be adjusted based on your layout requirements. Additionally, modern layout techniques like Flexbox or Grid are often preferred over floats for more complex layouts.

Q: Difference between background-size=100% and background-size=cover?

**Strings in Javascript:**

Strings in js are actually primitive and immutable. Primitive data types are actually immutable, meaning their value can not be changed after they have been created. Once a string is created there value can’t be changed. Any operation that appears to modify a string actually creates a new string with the modified value rather than altering the original value. This immutability ensures that string values remain consistent and predictable throughout the javascript program.

All string methods produce a new string without altering the original value. Following are the string methods:

* **Length:**

The length property returns the length of the given string in UTF-16 code               units;every code unit is exactly 16 bits long. The syntax of length property is:

**Str.length**

Here *str* is *string*.

The length property does not take any parameter.

**e.g:-**

Let str= “javascript”;

Let len = str.length;

console.log(len);            // 10

* **CharAt():**

The charAt returns the character at a specified index in a string. It returns a new string. CharAt() is similar to *bracket notation* to access a character at a specified index. The main differences are following:

* charAt() converts the index to integer while bracket notation does not, and directly uses index as a property name.
* charAt() returns **empty string** when index is out of range, while bracket notation returns **undefined.**

**e.g:**

const anyString = "Brave new world";

console.log(`The character at index 0   is '${anyString.charAt(0)}'`); //B

console.log(`The character at index 1   is '${anyString.charAt(1)}'`); // r

console.log(`The character at index 2   is '${anyString.charAt(2)}'`); // a

console.log(`The character at index 3   is '${anyString.charAt(3)}'`); //v

console.log(`The character at index 4   is '${anyString.charAt(4)}'`); //e

console.log(`The character at index 999 is '${anyString.charAt(999)}'`); // “”

* **CharCodeAt():**

CharCodeAt() returns the code of the character at the specified index in a string. This method returns the integer between **0-65535** representing the UTF-16 code unit at the given index. If the index is out of range **0-str.length-1** then it returns NaN. This method accepts one argument.

**e.g:**

Let greeting = “Good Morning”;

Let result = greeting.charCodeAt(5);

console.log(result);     //  109

* **At():-**

At() is introduced in ES2022. It also returns a character at a specified index. It is supported in all modern web browsers.This method allows for positive and negative integers. Negative integers count back from the last string character.

**e.g:**

const myString = "Every green bus drives fast.";

const atWay = myString.at(-2);

console.log(atWay); // 't'

* **Access [ ] :**-

It makes strings look like arrays(but they are not). If no character is found then it returns  **undefined ,**  while charAt() returns an empty string. It is just read-only, which means **str[0] = ‘A’**;  gives an error.

**e.g:-**

Let text= “Hello World”;

Text[0] = “A”;        // gives an error

* **Slice():-**

It extracts a part of the string and returns it in a new string. It takes two parameters: start position, end-position (end not included).

**e.g:-**

Let text = [‘apple’, ‘banana’, ‘kiwi’]

Let part = text.slice(7,13)             // banana

* **toUpperCase():-**

A string is converted to uppercase .

**e.g:-**

Let str = “Hello World”;

Let result = str.toUpperCase();

console.log(result);                     // HELLO WORLD

* **toLowerCase():-**

A string is converted to lowercase .

**e.g:-**

Let str = “Hello World”;

Let result = str.toLowerCase();

console.log(result);                   // hello world

* **concat():-**

It is used to join two or more strings.

**e.g:-**

Let str1 = “Hello”;

Let str2 = “World”;

Let result = str1.concat(“ ”, str2);

console.log(result);                       // Hello World

* **trim():-**

It is used to remove the white space from both sides of the string.

**e.g:-**

Let str = “      Hello World!       ;

Let result = str1.concat(“ ”, str2);

console.log(result);                     // Hello World!

* **trimStart():-**

It is used to remove the white spaces only from the start of the string.

**e.g:-**

Let str = “      Hello World!      ;

Let result = str1.concat(“ ”, str2);

console.log(result);                     // “HelloWorld!     “

* **trimEnd():-**

It is used to remove the white space only from the end of the string.

**e.g:-**

Let str = “      Hello World!       “;

Let result = str1.concat(“ ”, str2);

console.log(result);                     // “      Hello World!“

* padStart():

padStart() method pads a string to the start of the string. It pads with another string until it reaches a given length.

e.g:

Let text = ‘5’;

Let padded = text.padStart(4,’0’);

* padEnd():

padEnd() method pads a string from the end. It pads with another string until it reaches a given length.

e.g:

Let text = ‘5’;

Let padded = text.padStart(4,’0’);