CSS Interview Questions :-

Q① what is the purpose of CSS media queries?

CSS media queries are used to apply different styles to a webpage based on various characteristics of the user's device or viewpoint, such as its width, height, resolution, & even the type of device being used (eg screen, print, handheld devices) the purpose of media queries is to create responsive web designs that adapt to different screen sizes and devices, providing a better user experience across desktops, tables, smartphones & other devices.

Q② How do you write a media query in CSS?

Its a powerful tool that allows you to apply styles based on the device or viewport where a webpage is being displayed. Media queries enable you to create responsive designs that adapt to different screen sizes, resolutions & devices.

capabilities.

Syntax:- @media media-type and (media-feature) {

}

Q③ Explain the different max-width and min-width in media queries.

1. max-width:-

* This media feature specifies the maximum width of the viewport at which the style within the media query will be applied.

* Styles inside the media query will apply when the viewport width is equal to or less than the specified value.

* It is commonly used for creating styles that are applied to smaller viewport sizes, such as adjusting layout and typography for mobile devices.

2. min-width:-

* This media feature specifies the minimum width of the viewport at which the styles within the media query will be applied.

* Styles inside the media query will apply when the viewport width is equal to or greater than the specified value.

* It is commonly used for creating styles that are applied to larger viewport sizes, such as adjusting layout and spacing for desktop screens.

Q④ what is the purpose of the viewport meta tag in responsive web design?

The viewport meta tag is crucial in responsive web design as it controls how a webpage is displayed on a mobile device by defining the viewport's width and scale. Its purpose is to ensure that webpages render properly across various devices and screen sizes.

Q⑤ How can you apply different styles for landscape and portrait orientations using media queries?

① landscape orientation:- It has exactly the same uses the orientation media feature to switch between rows layout (in landscape) and a column layout (in portrait).

Ex:- HTML :- Box 1
 Box 2
 Box 3

CSS:-
```
body {
  display : flex;
}
div {
  background : yellow;
  width : 200px;
}
@ media (orientation : landscape) {
  body {
    flex-direction : row;
  }
}
```

(6) Explain the concept of a mobile-first approach in responsive design

① Mobile-first Styles:- CSS is styled targeting mobile devices first. This involves using media queries with a minimum width to apply additional styles for larger screens. Ex:-
```
body {
  font size : 16px;
}
@ media (min-width : 768px) {
  body {
    font-size : 18px;
  }
}
```

② Flexible layouts :- use percentage-based widths, flexible grid units like 'em' or 'rem' and CSS flexbox or grid layouts to create designs that adapt fluidly to different screen sizes

Q7) what are common breakpoints used in responsive design?

common breakpoints used in responsive design are specific viewport widths at which the layout of a webpage is adjusted to accommodate different screen size. It is defined using CSS media queries.

1. Extra Small (XS) :- This typically represents smartphones in portrait orientation. the breakpoint commonly used for xs is around 0 to 575 pixels.

2. Small (SM) :- This often represents smartphones in landscape orientation or small tablets. The breakpoint commonly used for SM is around 576 to 767 pixels.

3. Medium (MD) :- This is commonly represents ~~smartphones~~ large screens in landscape orientation or ~~small~~ large screens.

4. large (LG) :- This represents desktops and larger screens. The breakpoint commonly used for LG is around 992 to 1199 pixels.

5. Extra large (XL) :- This represents extra-large desktop screens

Q8 what is the purpose of the rem unit in media queries?
The rem unit in media queries serves the same purpose as in regular CSS stylesheets. It represents the font size of the root element (<html>), allowing for scalable and consistent designs across different viewport sizes.

Q9 How can you combine multiple media queries in CSS?
In CSS you can combine multiple media queries by chaining them together using logical operators such as and, or, and not. This allows you to create more complex conditions for applying styles based on various device characteristics.

Q10 what is the different significance of the all keyword in media queries?
In CSS media queries, the all keyword is used to target all types of media devices including screen, print, speech and other media types. It is often included implicitly if no specific media type is specified in the media query.

Syntax:- @media all and (max-width: 600px) {
}

Q(11) How do you use media queries to apply styles only for print stylesheets?

Syntax:- @media print {
/* Styles applied only when printing */
body {
font-family: Arial, sans-serif;
font-size: 12 pt;
color: #333
/* Add any other print-specific styles */
}
}

→ @media print {:- This media query targets the print media type, which is used when the webpage is being printed
→ within the curly braces { }, you can specify the CSS styles that should apply only when the page is being printed These styles might include adjustments to font sizes, colors, margins, hiding certain elements that are not relevant for printing, or other modifications to optimize the appearance of the printed page.

Q12) what is difference between screen and print in media queries?

1 Screen:-

* The screen media type is used to target styles for devices with a screen such as desktop monitors, laptops, tablets and smartphones
* Styles applied within a screen media query will affect how the webpage is displayed on any screen based devices
* This is the default media type if no media type is specified explicitly

2. print:-

* The print media type is used target styles for printing devices, such as printers or when a webpage is being printed to a physical medium
* Styles applied within a print media query will affect how the webpage appears when printed, allowing for optimization of the printing and printed output.
* when a webpage is printed, styles specified targeted for printing will be applied, while style targeted for screen based devices will be ignored

Q13) How can you hide an element on a specific screen size using media queries

To hide an element on a specific size using media queries, you can use the display property within a media query to set the elements display to none. For ex:-

```
@media (max-width: 767px) {
    .element-to-hide {
        display: none;
    }
}
```

* @media (max-width: 767 px):- This media query targets screens with a maximum width of 767 pixels, typically representing smaller devices such as smartphones & small tablets
* .element-to-hide - This is the selector for the element you want to hide.
* display: none;:- This CSS property hides the element by removing it from the document flow, making it invisible and not taking up any space on the webpage.

Q(14) Explain the role of the orientation property in media queries.

The orientation property in media queries allows you to target specific devices orientations, such as portrait or landscape

How the orientation property works in media queries:-

1. portrait Orientation:-
* when the viewport height is greater than or equal to its width, the device is in portrait orientation

* This is commonly seen on smartphones and smaller tablets held in a vertical position

* you can use the orientation: portrait condition in media queries to target styles specifically for portrait orientation

2. landscape Orientation:-
* when the viewport width is greater than its height, the device is in landscape orientation.

* This is typically observed on devices such as tablets and desktop monitors, or smartphones held horizontally.

* you can use the orientation: landscape condition in media queries to target styles specifically for landscape orientation.

(15) How do you target specific devices using media queries?

Targeting specific devices using media queries involves using combinations of media features such as screen size, resolution, and aspect ratio to match the characteristics of the target devices.

-> Here are general approach to target specific devices using media queries

① Identify target Devices:- start by identifying the specific devices or categories of devices you want to target, laptops or desktops.

② use media features:- like min-width, max-width, min-height, max-height, orientation

③ Test and Refine:- Test your media queries on various devices to ensure they behave as expected.

④ Adjust as Needed:- Adjust your media queries based on feedback and testing results.

Q16. what is the purpose of the not keyword in media queries?

In CSS media queries, the not keyword is used to negate a condition, allowing you to apply styles when a specific condition is not met. It is useful for targeting devices or conditions that do not match a certain criteria.

Q17. How can you use media queries to adjust font sizes for different screen sizes?

Media queries to adjust font sizes for different screen sizes by defining specific font sizes within different media query range.

Ex:-
```
/* Default font size */
body {
  font-size: 16px;
}

/* Media query for smaller screens */
@media (max-width: 768px) {
body {
  font-size: 14px;
}
}
```

Q18. what is the box-sizing property in CSS and what does it control?

The box-sizing property in CSS controls how the width and height of elements are calculated, specifically in relation to the content, padding and border.

There are two main values for the box-sizing property

① content-box (default):-
* This is the default value for the box-sizing property.
* with box-sizing: content-box, the width and height of an element are calculated based only on the content's width and height.
* using content-box, the specified width and height of an element only include the content area.

② Border-box:- * with box-sizing: border-box, the width and height of an element include the contents width and height, as well as any padding or border added to the element.
* using border-box, the specified width and height of an element include the content area, padding and border and any changes to padding.

Ex:- /* content-box */
.element-content-box {
box-sizing: content-box;
width: 200 Px;
Padding: 20 Px;
border: 2 Px solid black;
}

/* Border-box */
.element-border-box {
box-sizing: border-box;
width: 280 px;
padding: 20 px;
border: 2 Px solid black;
}

Q(19) Explain the difference between box-sizing content-box; and box-sizing: border-box;

① content-box:-
* This is the default value for the box-sizing property.

* with box-sizing: content-box, the width and height of an element are calculated based only on the contents width and height, excluding any padding or border added to the element.

② border-box:-
* with box-sizing: border-box, the width and height of an element include the contents width and height, as well as any padding & border added to the element.
* It specified width and height of an element only include the content area, and any padding & border do not affect the overall width and height of the element.

Q(22) Difference between normalizing and resetting

Resetting	Normalizing
① Resets all the styles that comes with the browser's user agent.	① provides cross-browser consistency in the default styling of HTML elements which are provided by the browser's user agent.
② It is hard to debug as it is nearly impossible to identify bugs.	② Debugging is easy while normalizing.
③ Resetting is Non Modular (no section breaking in styles)	③ Normalizing is Modular: (styling is divided into sections for ease)

Q23 what is a CSS combinator, and how it is used in a selector?

In CSS, a combinator is a special character that specifies the relationship between two selectors, indicating how they are related to each other and how the styles should be applied based on that relationship.
There are four main types of combinators in CSS:
① Descendant combinator (whitespace)
② child combinator (>)
③ Adjacent sibling combinator (+)
④ General sibling combinator (~)

Q24 Differentiate between descendant and child combinators in CSS selectors. Provide example for each

Descendant (whitespace)	child Combinator (>)
① The descendant combinator selects an element that is a descendant of another element, regardless of how deeply nested it is in the HTML structure.	① The child combinator selects an element that is a direct child of another element, meaning it is an immediate child and not further nested.

② It is represented by whitespace b/w two selectors
Ex:- div span {
/* styles */
}
	② It is represented by the > symbol
	ex:-
	div > span {
	/* style */
	}

Q25 Explain the purpose of the adjacent sibling combinator (+) in CSS. Provide a use case

The adjacent sibling combinator (+) in CSS is used to select an element that is immediately preceded by another specific element, and they share the same parent in the HTML structure.
Ex:- HTML
 item 1
 item 2
<li class = "highlight"> item 3
 item 4

.highlight + li {
color : red;
}

Q(26) How does the general sibling combinator (~) differ from the adjacent sibling combinator (+)?

The general sibling combinator ('~') and the adjacent sibling combinator ('+') in CSS are both used to select sibling elements based on their relationship to another specific element in the HTML structure.

① Adjacent sibling Combinator ('+')
- The adjacent sibling combinator selects an element that is immediately preceded by another specific element; and they

Ex:- "CSS"
```
div + p {
  /* styles */
}
```
- - -

② General sibling combinator ('~') :-
The general sibling combinator selects elements that are immediately adjacent & not.

Ex:- "CSS"
```
div ~ p {
  /* styles */
}
```

Q(27) what is the significance of the child combinator (>) in CSS selectors?

The significance of the child combinator lies in its ability to provide a high level of specificity in selecting elements based on their direct parent-child relationship in the HTML structure.

Ex:- "HTML"
```
<div class="container">
  <ul>
    <li> item 1 </li>
    <li> item 2 </li>
    <li> item 3 </li>
  </ul>
  <p> paragraph </p>
</div>
```

→ "CSS"
```
/* styles <li> elements that are direct children
   of <ul> elements */
ul > li {
  /* styles */
}
```

Q29) provide an example of using the descendant combinator to style nested elements.

ex:- "HTML"

```
<div class="container">
<h2> Title </h2>
<ul>
<li> item 1 </li>
<li> item 2 </li>
</ul>
</div>
```

"CSS" =

```
.container h2{
    color: blue; /* Styles applied to h2 elements */
}
.container ul li{
color: green;
}
```

→ container h2 :- Selects <h2> elements that are descendants of elements with the class container and applies a blue color to their text.

→ .container ul li :- Selects <li> elements that are descendants of <ul> elements which, in turn, are descendants of elements with the class 'container'.

Q30) Explain how the space between two selectors represents a descendant combinator.

The space between two selectors represents the descendant combinator. It indicates that the second selector is a descendant of the first selector, meaning it is nested within the first selector in the HTML structure.

ex:- "html"

```
<div class="container">
<h2> Title </h2>
<ul>
<li> item 1 </li>
<li> item 2 </li>
</ul>
</div>
```

"CSS"

```
.container h2{
/* Styles */
}
```

Q31) How would you select an element that is the immediate next sibling of another element in CSS?

To select an element that is the immediate next sibling of another element in CSS, we can use adjacent sibling combinator, represented by the + symbol.

Ex:- CSS:-

```
/* Selects the <p> element that is the immediate next sibling of a <div> */
div + p {
  /* Style */
}
```

→ div :- Selects the ' ' element

→ '+' :- Represents the adjacent sibling combinator.

→ 'p' Selects the <p> element that is the immediate next sibling of the .

Q(32) In what scenarios would you choose one combinator over another, and what are the considerations when using combinators for efficient CSS selectors?

There are four different combinators in CSS:-

(1) descendant selector (space)
(2) child selector (>)
(3) adjacent sibling selector (+)
(4) general sibling selector (~)

(1) Descendant selector:- It matches all elements that are descendants of a specified elements.

Eg:- div p {
  bag background-color: yellow;
}

(2) child selector (>) :- It selects all elements that are the children of a specified element.

Eg:- div > p {
  background-color: yellow;
}

(3) Adjacent sibling selector (+):- it is used to select an element that is directly after another specific element.

Q(33) Explain the concept of CSS pseudo-selectors. Provide examples of commonly used pseudo-selectors and their purposes.

CSS pseudo-selectors are special keywords that allow you to select elements based on their state or position in the document tree; pseudo-selectors begin with a colon (:) followed by the keyword representing the state or position of the element.

(1) :hover :-

- purpose :- Selects an element when the user hovers over it with the mouse cursor

Ex:- CSS :- button:hover {
background-color: #ff0000;
}

② :active :- Selects an element while it is being activated by the user, typically while clicking.
Ex:- CSS :- button:active {
transform: translateY(2px);
}

③ :focus :- Selects an element when it receive focus, typically via keyboard navigation or when clicked. Ex:- CSS :- input:focus {
border-color: #00ff00;
}

④ :first-child :- Selects the first child element of its parent.
Ex:- CSS :- li:first-child {
font-weight: bold;
}

⑤ :last-child :- Selects the last child element of its parent.
Ex:- CSS :- div p:last-child {
color: #0000ff;
}

Q(34) Differentiate between pseudo-classes and pseudo-elements in CSS. Give examples of each.

① pseudo-classes :-
- pseudo-classes are used to defined the special state of an element, such as its interaction state or its position within the document tree.
- pseudo-classes are prefixed with a colon (':') and come after the selector.
- Ex:- includes ':hover', ':active', ':nth-child()'
- Eg:- CSS :- a:hover {
color: red;
}

② pseudo-elements :-
- pseudo-elements are used to style specific parts of an element's content, such as the first letter line of text.
- pseudo-elements are prefixed with two colons ('::')
Eg:- '::first-letter', '::first-line'.
Ex:- /* styles the first letter of a paragraph */
p::first-letter {
font-size: 150%;
}

(35) How can you use the :nth-child pseudo-class to select specific elements in a list or container?

The ':nth-child()' pseudo-class in CSS allows you to select elements based on their position relative to their parent container. you can specify a formula within the parentheses to target specific elements.

Ex:- 'HTML'

```
<ul>
<li> item 1 </li>
<li> item 2 </li>
<li> item 3 </li>
<li> item 4 </li>
</ul>
```

'CSS'

```
li: nth-child(odd){
 background-color : #f0f0f0;
}
/* Style the even-numbered <li> elements */
li: nth-child (even){
background-color : #e0e0e0;
}
```

JAVA SCRIPT Interview Questions

Q① what are the primitive data types in javascript?

① Number :- Represents numeric data, including integers and floating-point numbers.

Eg:- let weight = 5.7
console.log(weight)

② String :- Represents a sequence of characters enclosed in single & double quotes.

Eg:- let b = 'Shikha'
let substr = b.substr(0,3)
console.log(substr)

let substring = b.substring(0,4)
console.log(substring)

③ Boolean :- Represents a logical value true & false.

Eg:- let something5 = true
let something6 = false
console.log(10==20)

④ undefined :- Represents a variable that has been declared but not assigned a value.

Eg:- let something.
let something 2 = undefined
let something 3 = "India".
let x = 200

⑤ Null :- Represents the intentional absence of any value.

Eg:- let some1 = null

Q② what is the difference between null & undefined in javascript

undefined	Null
① undefined means a variable has been declared but has yet not been assigned a value	① Null is an assigned value. It be assigned to representation of no value
② It is a global property	② It is not a global property.

Ex:- let x;
console.log(x);
function doSomething(){
}
console.log(doSomething());
	Ex:-
	let y = null
	console.log(y);

Q③ How do you check the data type of a variables in javascript?

We can check the data type of a variable using the 'type of' operator. The 'type of' operator returns a string indicating the data type of the operand.

Eg:- let x = 10;
console.log (typeof x);

let y = "Hello".
console.log (typeof y);

Q④ what is the difference between '==' (equality) and '===' (strict equality) operators in javascript, and how do they relate to data types?

'=='	'==='
① == is used for comparing two variables, but it ignores the datatype of variables	① === is used of comparing two variables, but this operator also checks datatypes and compares two values
② checks the equality of two operands without considering their types	② Compares equality of two operands with their types.

⑤ Return true if the two operands are equal. It will return false if the two operands are not equal.

Ex:- console.log(5=="5");
console.log(0==false);
console.log(null==undefined);

③ It returns true if both values and data types are the same for the two variables.

④ Ex:-
console.log(5===5);
console.log(0===false);
console.log(null===undefined);

Q⑥ How do you convert a string to a number in JavaScript?

① using the Number() function ***;
The Number() function converts its argument to a number data type.
Eg:- let str = "123";
let num = Number(str);
console.log(num);

② using the parseInt() function:-
The parseInt() function parses a string argument and returns an integer of the specified radix
Eg:- let str = "123";
let num = parseInt(str);
console.log(num);

⑤ parseFloat() function
The parseFloat() function parses a string argument & returns a floating-point number.
Eg:- let str = "123.45";
let num = parseFloat(str);
console.log(num);

Q⑦ Explain the difference between the ++x and x++ increment operators in JavaScript.

① The ++x (pre-increment):-
'++x' operator increments the value of 'x' before returning the incremented value.
Eg:- let x = 5;
let y = ++x;

② x++ (post-increment):-
The x++ operator increments the value of 'x' but returns the original value of x before the increment.
Eg:- let x = 5;
let y = x++;

Q⑧ How does javascript handle NaN (NOT a Number) values, and how you check if a value is NaN?

A

To check if a value is 'NaN', you can use the 'isNaN()' function or the 'Number.isNaN()' method.

① isNaN() function :-
The isNaN() function returns 'true' if the given value is 'NaN' & 'false' otherwise.

Eg :- isNaN(123); // false

② Number.isNaN() Method :-
The 'Number.isNaN()' method returns 'true' only if the value is exactly 'NaN', and 'false' for any other value.

Eg :- Number.isNaN(NaN); // true
Number.isNaN(123); // false
Number.isNaN('hello'); // false

Q⑩ what is the purpose of the undefined data type, and when might it be explicitly assigned to a variable?

- The undefined data type in javascript represents a variable that has been declared but not assigned a value.

① Variable declaration without initialization

```
let x;
console.log(x);
```

② function parameters without arguments

```
function foo(parameter){
  console.log(parameter);
}
foo();
```

③ Explicit assignment

```
let y = undefined;
console.log(y);
```

Q(12) what is Hoisting?

Hoisting is a machanism where variable and function declarations are moved to the top of their containing scope during the compile phase, befor the code is executed.

① variable Hoisting:-
variable declarations (but not initizations) are hoisted to the top of their containing scope.

Eg:- console.log(x);
var x=5;

② function Hoisting:-
function declarations (not function expressions) are also hoisted to the top of their containing scope.

Eg:- greet();
function greet(){
console.log("Hello");
}

Q(13) what is IIFE?

IIFE stands for Immediately Invoked function Esxpression. It's common design pattern in javaScript where a function is defined and immediately executed within its lexical scope.

Eg:- (function(){
var x=10;
console.log(x); //output 10
})();

Q(14) what is meant by Default parameter passing.

The default parameter is a way to set default values for function parameters a value is not passed in (i.e it is undefined).

Eg:- function test(num1, num2=num1*2){
console.log(num1 * num2);
}
test(2);
test(2,3);

Q(15) what is the default return value
If a function does not explicitly return a value, it automatically returns 'undefined'. This default return value applies to functions that reach the end of their execution without encountering a 'return' statement, as well as to functions that have an empty 'return' statement.

```
Eg:- function doSomethingElse(){
       return;
     }
     const result = doSomethingElse();
     console.log(result);
```

Q(16) How to pass unlimited number of parameters to a function

we can pass an unlimited number of parameters to a function in javascript by using the rest parameter syntax. rest parameters allow you to represent an indefinite number of arguments as an array, making it easy to work with variable-length argument lists within a function

```
Eg:- function concatStrings(separator, ...strings){
       return strings.join(separator);
     }
     console.log(concatStrings('-','a','c'))
     console.log(concatStrings('-','x','y','z'));
```