CSS Interview questions

1. What is the purpose of CSS media queries?

→ The media query in CSS is used to create a responsive web design. They allow us to create different layouts depending on the size of the viewport.

Syntax: @media <media-type> and (media-feature-rule) { }

media-type values could be all, print, screen, etc

2. How do you write a basic media query in CSS.

→ @media media-type and (media-feature-rule) {
/* CSS rule goes here */
}

→ A media type tells the browser what kind of media the code is for (eg. print or screen).

→ A media expression, which is a rule, or test that must be passed for contained CSS to be applied

→ A set of rules that will be applied if the test passes and the media type is correct.

3. Explain the difference b/w max-width & min-width in media queries.

→ max-width: styles are applied if the browser ~~size~~ window is less than or equal to certain width.

(min-width: 1000px) : styles are applied, if the browser window is >= 1000px.

4. What is the purpose of viewport meta-tag in responsive web design?

→ Setting the viewport meta tag lets us control the width & scaling of the viewport so that it is sized correctly on all devices.

5. How can you apply different styles for landscape & portrait orientations using media queries?

→ By using the "orientation" media feature rule.

ex:
```
@media screen and (orientation: landscape){
    body {
      color: red;
    }
}
@media screen and (orientation: portrait){
    body{
      color: blue;
    }
}
```

6. Explain the concept of mobile first approach in responsive design.

→ Mobile-first approach is designing the smallest screen first & gradually working up to larger screen sizes.

7. What are common breakpoints used in responsive design?

→ 320px - 480px : Mobile devices
→ 481px - 768px : iPads, Tablets
→ 769px - 1024px : small screens, Laptops
→ 1025px - 1200px : Desktop, Large screens
→ 1201px & more : Extra Large screens, TV.

8. What is the purpose of the rem units in media queries.

→ rem units allows to specify sizes that are relative to the root element font size, rather than specifying sizes in absolute units like px.

9. How can you combine multiple media queries in CSS?

→ we can combine multiple media queries into a single rule by separating them with commas

ex: @media print, screen { }

10. What is the significance of all keyword in media queries?

→ all means suitable for all kinds of devices

11. How do you use media queries to apply styles only for print stylesheets?

→ By using "print" media-type.

ex: @media print {
body{ color: blue;}
}

12. What is the difference b/w screen & print in media queries?

→ screen styles screens like mobiles, tablet, laptop
→ print works for printable pages.

13. How can you hide an element on a specific screen size using media queries?

→ @media screen and ~~(min-width)~~ (max-width: 500px){

```
.header{
  display: none;
}
}
```

The above code will hide the element when the viewport size is ≤500 px.

14. Explain the role of orientation property in media queries.

→ It is used to change↑ or apply styles depending on the orientation of the browser.
→ orientation takes values like landscape, portrait.

15. How do you target specific devices using media queries?

→ In media queries, logical operators like and, not, and only are used to target specific devices.
→ For ex. @media only screen and (max-width - 1000px){} targets only screens smaller than 1000 px.

16. What is the purpose of the not keyword in media queries?

→ Not negates / reverses the meaning of the entire media query.

```
@media not all and (orientation : landscape){
    body{
        color : blue;
    }
}
```

The above example changes the text-color to blue only if the orientation is portrait.

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

→ By specifying the font-sizes for different devices we use different breakpoints.

```
body {
  font-size: 10px;
}
@media (min-width: 200px) and (max-width: 500px) {
    body{
    font-size: 16px;
    }
}
@media (min-width: 500px) {
  body{
  font-size: 32px;
```

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

→ It specifies how to calculate the total width & height of an element.

→ When box-sizing is set to content-box. The total width would be (width + 2*padding + 2*borders)

→ When it is borders-box. The total width would be equal to the width of that element.

19. Explain the difference b/w box-sizing: content-box; & box-sizing: borders-box;

content-box

→ Default value to the box-sizing property

→ The width & height properties include the content, but does not include the padding, borders, or margin.

borders-box

→ The width & height properties include the content, padding & borders, but do not include the margin.

20. How does the box-sizing property affect the calculation of an element's width & height in CSS?

ex:
```
.box {
    width: 350px;
    borders: 10px solid black;
}
```

When box-sizing: content-box;
width = width of the content (borders, padding are not included). total width = 350+20 = 370px

box-sizing: borders-box;
width = borders + padding + width of the content.
total width = 350 px.
330 (content width) + 20 (borders width)

21. Why might you choose box-sizing property to be border-box as default box-model for your project?
→ It makes easier to size elements in the HTML page.

22. Difference b/w normalizing & resetting.
→ CSS reset removes all built-in browser styling. elements like h1-h6, p, strong etc looks exactly similar. we are then supposed to add styling to the elements however we need
→ Normalize. makes the built-in browser styling consistent across browsers. elements like h1-h6 appear bold, larger. in a consistent way across browsers.

23. What is a CSS combinator, & how is it used in a selector?
→ combinators define the relationship b/w selectors.
→ Using combinators, we combine selectors to select elements based on their relationship with ~~other~~ other elements.
→ Ex: Descendant combinator, child combinator, adjacent-sibling, next-sibling combinator, etc

24. Difference b/w descendant & child combinators in CSS selectors. Provide ex for each

Descendant selector	child selector
→ It is used to select all the elements which are child of the element.	→ It is used to select all the direct children of the element.
ex: element element { }	ex: parent > child

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

→ + selector is used to select an element that is immediately after another specific element.

→ sibling elements must have the same parent element.

ex: div + p {
  color: red;
}

⇒ It targets all the 'p' element that is immediately after 'div' element, & changes the text color to red.

26. How does the general sibling selector (~) differ from adjacent sibling (+) selector?

→ div ~ p : selects all 'p' elements that comes after the 'div' element, these 'p' elements need not be immediate element of 'div'.

→ div + p : It selects only the 'p' tags that are immediately after 'div' tag.

27. What is the significance of (>) combinators?

→ It is used to select the direct children of an element. It won't select the grand children like descendant selectors.

28. How can you select all paragraphs that are direct descendants of a div using CSS selectors?

→ div > p

29. Provide an example of using the descendant combinator to style nested elements.

```
<section>
    <article>
        <p> Lorem </p>
        <small> text </small>
    </article>
    <footer> footer </footer>
</section>
```

```
section * {
    opacity : 0.5
}
```

⇒ sets the opacity of all the elements that are inside the section to 0.5.

30. Explain how the space between two selectors represents as descendant selector:

→ A space is used to select elements nested within other elements. For ex, the following CSS rule selects only paragraph elements that are located inside a div.

ex: div p { }

31. How do you select an element that is the immediate next sibling of another element in CSS?

→ By using (+) combinator.

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

→ When we want to target all the immediate children, we use (>) combinator.

→ When we want to target all the descendants, we use space.

→ When we want to target a element based on the previous sibling we use (~) or (+) combinator.

33. Explain the concept of CSS pseudo-selectors. Provide ex of commonly used pseudo-selectors & their purpose.

→ pseudo selectors are used to target an element based on its state or its position among its siblings in a document.

:hover → targets an element when it is hovered.

:visited → targets when a link is visited.

:link → matches links that have not yet been visited.

:nth-child → select elements from a list of sibling elements.

:nth-of-type → select elements from a list of sibling elements that match a certain type from a list of sibling elements.

34. Difference b/w pseudo-classes & pseudo-elements in CSS

pseudo-classes	Pseudo-elements
→ are based on state or user interaction on the element	→ styles the specific part of an element
→ starts with :name	→ starts with ::name
ex: p:hover { color: red; }	ex: p::first-letter { font-size: 32px; }

35. How can you use the :nth-child pseudo class to select specific elements in a list or container? Provide an ex

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

li:nth-child(even){} selects (2)
li:nth-child(odd){} selects (1 & 3)
li:nth-child(n+2){} selects (2 & 3)

ARUN'S

JS Interview questions

1. What are the primitive data types in JS?

Primitive data-types can store only a single value & these are immutable.
In JS, we have 7 types : number, string, boolean, undefined, symbol, null

2. Explain the difference b/w null & undefined?

→ when a variable is declared & not assigned any value, then it will have 'undefined' assigned to it by default.
→ null represents the absence of value, we need to explicitly assign null to a variable.

3. How do you check the datatype of a variable in JS?

→ Using "typeof".

4. Explain the concept of truthy & falsy values in JS. provide ex:

→ In JS, values like 0, null, undefined, "", NaN, false. Apart from these everything is a truthy value.
→ when we pass the value to Boolean(), if the value lies in falsy values, then it will return false, else returns true.

5. What is the difference b/w == & === operators in JS. & how do they relate to data types?

→ x ===y checks for both data type & value
→ x==y, checks if the value of x & y are same or not.

6. How do you convert a string to a number in JS?

→ By using Number(), parseInt(), parseFloat().

→ By using unary operator (+).

7. Explain the difference b/w ++x & x++ in JS?

```
Let x = 5;
console.log(x++); // first prints 5, then increments
                  the x value by 1.
conole.log(++x); // first increments the x value
                  by one, the prints 6.
```

8. How does JS handle NaN values, how can you check if a value is NaN?

→ when we try to perform a illegal arithmetic operation, then the result would result in NaN.

→ for ex 1+2 // 3

1+"abc" // NaN

→ we can check if a value is NaN or not by using Number.isNaN() method.

9. Explain the concept of type coersion in JS. Provide example

→ Coersion is the process of converting one datatype to other

→ JS implicitly coerses when adding two values, x+y

if x is number & y is string

x is converted to string & concatenated with y

10. what is the purpose of the undefined datatype, & when it be explicitly assigned to a variable?

→ purpose of undefined datatype is to denote an absence of meaningful value.

→ It when a variable is declared but has not yet been assigned a value, then undefined would be its default value.

11. How do you create and use template literals in JS?

→ It can be created by enclosing a text in a backticks (``)

→ with template letexals, we can embed a expression in a string

ex: let a=1;
let b=2;
console.log(`sum is ${a+b}`)
output : sum is 3

12. what is Hoisting?

→ Hoisting allows us to access variables declared using var, & functions (normal) before its declaration

ex: console.log(a); // undefined
var a=5;
f()
function f(){
console.log("hi");
}

// "hi"

13. what is IIFE ?

→ IIFE is a JS function that runs as soon as it is defined.

syntax: (function(){ (()=>{})();
// ... (or)
})();

14. what is meant by default parameter passing?

→ Default parameters allows named parameters to be initialized with default value if no value or undefined is passed.

```
function sum(a, b=0){
    return a+b;
}
sum(5,6) // 11
sum(5) // 5
```

15. what is the default return value?

→ "undefined"

16. How do you pass unlimited no. of parameters to a function?

→ Using the rest operator. (...)

→ Using arguments keyword (does not work for arrow function)