Interview questions and answers

HTML AND CSS

1. What are tags and attributes in HTML?

Tags are the primary component of the HTML that defines how the content will be structured/ formatted, whereas Attributes are used along with the HTML tags to define the characteristics of the element. For example, <p align=" center">Interview questions</p>, in this the 'align' is the attribute using which we will align the paragraph to show in the center of the view.

2. What are void elements in HTML?

HTML elements which do not have closing tags or do not need to be closed are Void elements. For Example <br />, <img />, <hr />, etc.

3. Are the HTML tags and elements the same thing?

No, HTML tags are used to define the structure of a web page, while HTML elements are made up of a set of tags that define a specific part of a web page.

4. What is the advantage of collapsing white space?

In HTML, a blank sequence of whitespace characters is treated as a single space character, Because the browser collapses multiple spaces into a single space character and this helps a developer to indent lines of text without worrying about multiple spaces and maintain readability and understandability of HTML codes.

5. What are HTML Entities?

In HTML some characters are reserved like '<', '>', '/', etc. To use these characters in our webpage we need to use the character entities called HTML Entities. For example : To display a less than sign (<) we must write: &lt; or &#60;

6. How do you create nested web pages in HTML?

Nested web pages basically mean a webpage within a webpage. We can create nested web pages in HTML using the built-in iframe tag. The HTML <iframe> tag defines an inline frame. For example:

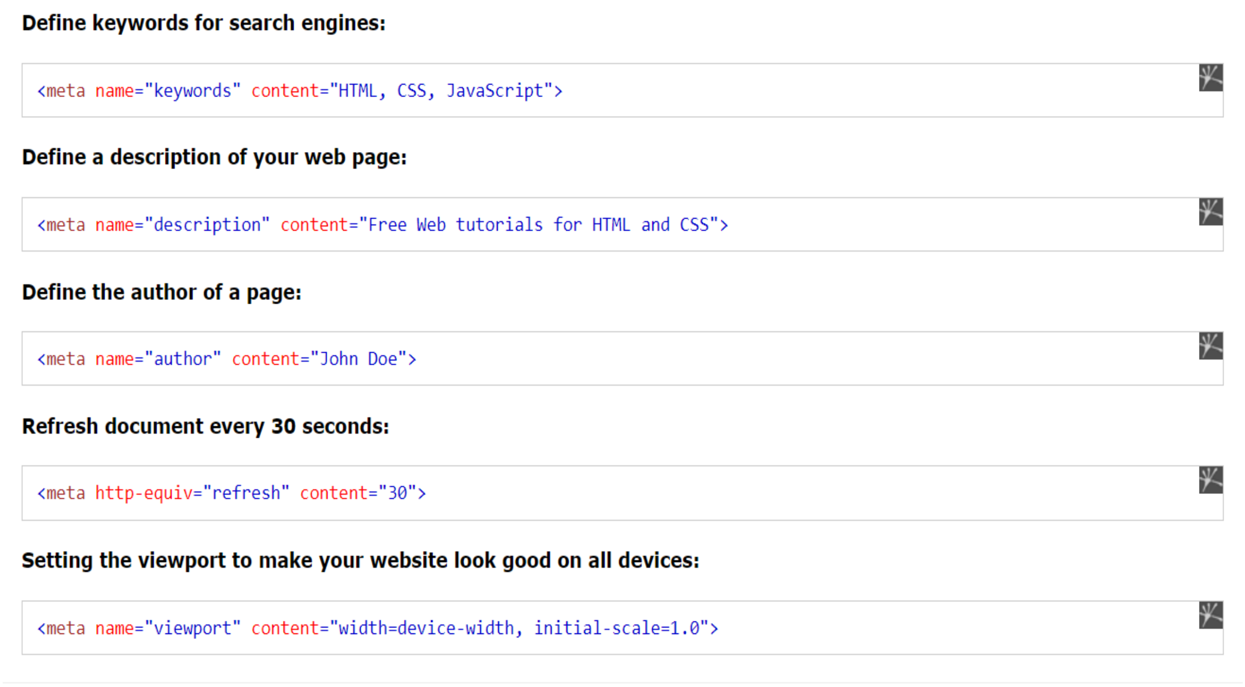
```
<!DOCTYPE html>

<html>

  <body>

    <h2>HTML Iframes example</h2>

    <p>

      specify the size of the iframe using the height and width attributes:

    </p>

    <iframe src="https://abc.com/" height="600" width="800"></iframe>

  </body>

</html>
```

7. How meta tags is used in HTML file and what are its functionalities

The <meta> tag defines metadata about an HTML document. Metadata is data (information) about data.

<meta> tags always go inside the <head> element, and are typically used to specify character set, page description, keywords, author of the document, and viewport settings.

Metadata will not be displayed on the page, but is machine parsable.Metadata is used by browsers (how to display content or reload page), search engines (keywords), and other web services.



Define keywords for search engines:

```
<meta name="keywords" content="HTML, CSS, JavaScript">
```

Define a description of your web page:

```
<meta name="description" content="Free Web tutorials for HTML and CSS">
```

Define the author of a page:

```
<meta name="author" content="John Doe">
```

Refresh document every 30 seconds:

```
<meta http-equiv="refresh" content="30">
```

Setting the viewport to make your website look good on all devices:

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

8. What is the difference between the 'id' attribute and the 'class' attribute of HTML elements?

Multiple elements in HTML can have the same class value, whereas a value of id attribute of one element cannot be associated with another HTML element.

9. Explain CSS position property?

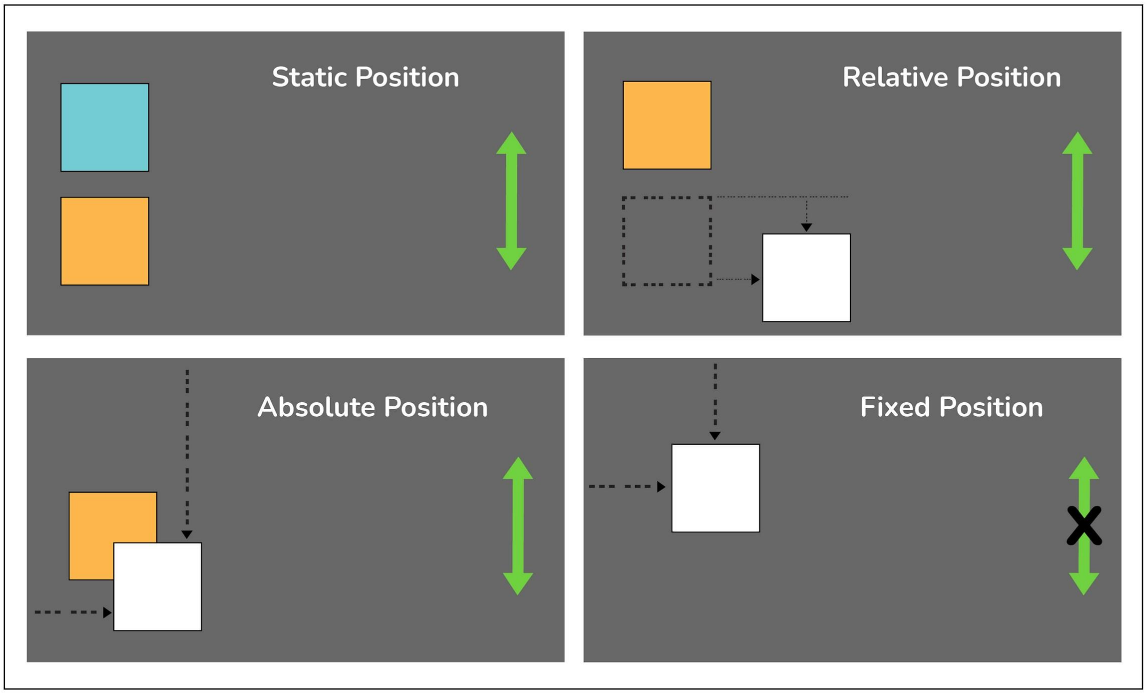
Absolute: To place an element exactly where you want to place it. absolute position is actually set relative to the element's parent. if no parent is available then the relative place to the page itself (it will default all the way back up to the element).

Relative: "Relative to itself". Setting position: relative; on an element and no other positioning attributes, it will no effect on its positioning. It allows the use of z-index on the element and it limits the scope of absolutely positioned child elements. Any child element will be absolutely positioned within that block.

Fixed: The element is positioned relative to the viewport or the browser window itself. viewport doesn't change if you scroll and hence the fixed element will stay right in the same position.

Static: Static default for every single page element. The only reason you would ever set an element to position: static is to forcefully remove some positioning that got applied to an element outside of your control.

Sticky: Sticky positioning is a hybrid of relative and fixed positioning. The element is treated as relative positioned until it crosses a specified threshold, at which point it is treated as fixed positioned.



InterviewBit

10. What is the difference between “display: none” and “visibility: hidden”, when used as attributes to the HTML element.

When we use the attribute “visibility: hidden” for an HTML element then that element will be hidden from the webpage but still takes up space. Whereas, if we use the “display: none” attribute for an HTML element then the element will be hidden, and also it won’t take up any space on the webpage.

11. What is the difference between inline and block elements

1. Block-Level Elements: Block-level elements take up the full width available on their own line, effectively creating a "block". They always start on a new line in the document. Even if the actual content doesn't fill the whole width, the block space still remains reserved, and no other element can appear next to it on the same line. The width and height properties are respected for block elements.

Examples of block elements include <div>, <p>, <h1>-<h6>, <ol>, <ul>, <li>, <header>, <footer>, <section>, and many more.

2. Inline Elements: On the contrary, inline elements only take up as much width as necessary based on their content, and they do not start on a new line. They align themselves on the same line as their neighboring elements if there's space available. The width and height properties do not apply to inline elements.

Examples of inline elements include <span>, <a>, <img>, <em>, <strong>, <br>, and more.

In addition, there's a hybrid of these two called inline-block. An inline-block element is displayed inline, just like an inline element, but it also behaves like a block element in terms of accepting width and height values. This can be especially useful when you want to place elements side-by-side (like inline elements) but also want to control their dimensions (like block elements).

12. What is form tag in HTML and what attributes does it have?

The <form> tag in HTML is used to create an HTML form for user input. The <form> element can contain one or more of the following form elements:

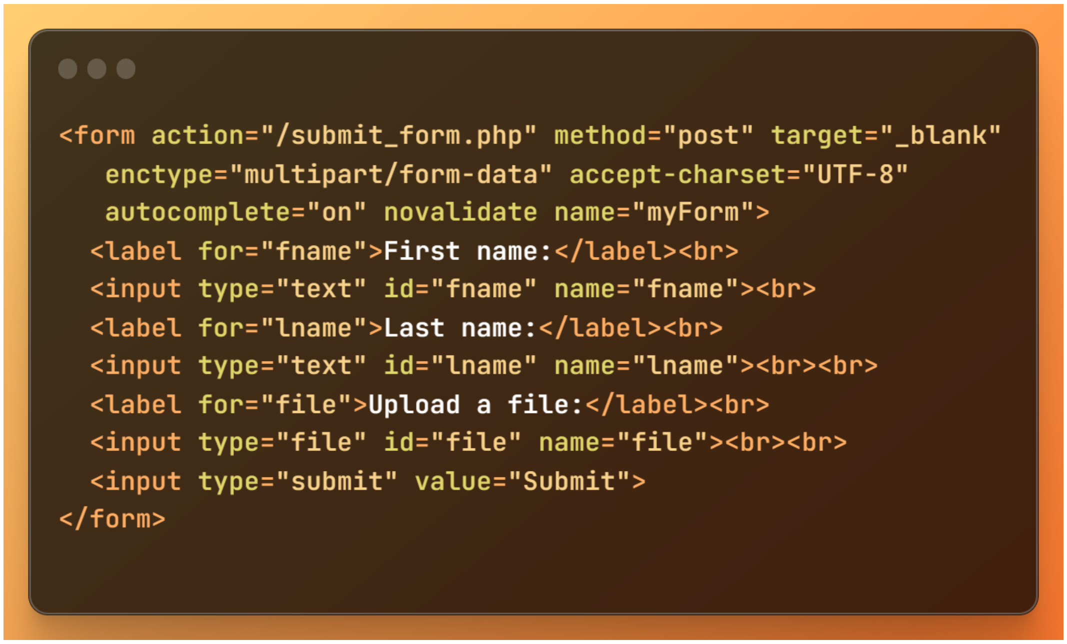
- <input>
- <textarea>
- <button>
- <select>
- <option>
- <optgroup>
- <fieldset>
- <label>

A <form> tag can have several attributes. As of my knowledge cutoff in September 2021, here are some of the main attributes that can be included in a <form> tag:

- action: Specifies the URL of the page that will process the input from the form (the form handler). This page can be a PHP, ASP, etc. page, which is capable of processing form data.

- method: Specifies the HTTP method to use when sending form data. The two most common methods are GET and POST.
- target: Specifies where to display the response that is received after submitting the form. The different targets are _blank (opens the linked document in a new window or tab), _self (opens the linked document in the same window/tab as it was clicked, which is the default), _parent (opens the linked document in the parent frame), and _top (opens the linked document in the full body of the window).
- enctype: Specifies the type of content that is used to submit the form to the server. The possible values are application/x-www-form-urlencoded (which is the default), multipart/form-data (used for binary data and file uploads), and text/plain.
- accept-charset: Specifies the character encodings that are to be used for the form submission.
- autocomplete: Specifies whether a form should have autocomplete on or off.
- novalidate: This boolean attribute indicates that the form should not be validated when submitted.
- name: Specifies the name of the form.

Example:



```
<form action="/submit_form.php" method="post" target="_blank"
   enctype="multipart/form-data" accept-charset="UTF-8"
   autocomplete="on" novalidate name="myForm">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname"><br><br>
  <label for="file">Upload a file:</label><br>
  <input type="file" id="file" name="file"><br><br>
  <input type="submit" value="Submit">
</form>
```

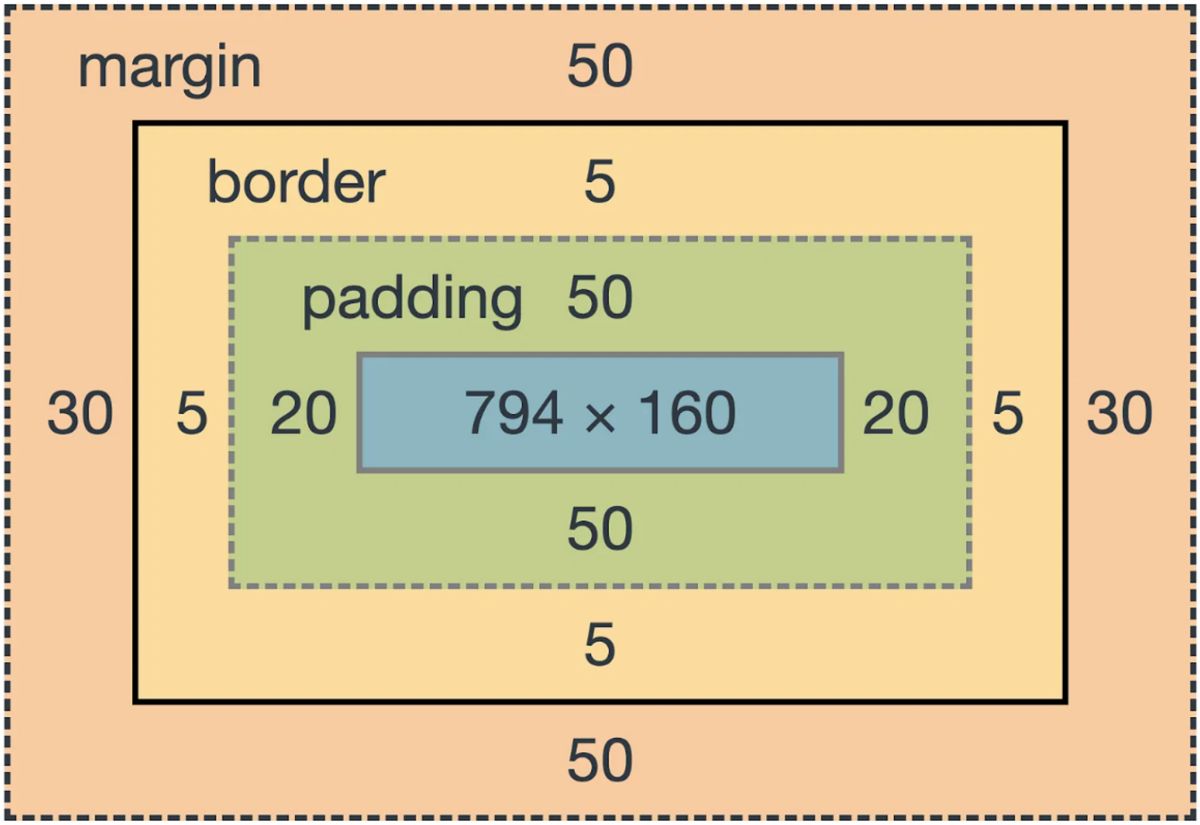
Let's break down the example:

- action="/submit_form.php": This form data will be sent to submit_form.php on submission.
- method="post": The form data will be sent via HTTP POST method.

- target="_blank": The response will be displayed in a new tab or window.
- enctype="multipart/form-data": This is necessary for file uploads.
- accept-charset="UTF-8": The form submission will be encoded in UTF-8.
- autocomplete="on": Browsers will attempt to autocomplete the form inputs based on past user input.
- novalidate: The form will not be validated when it is submitted. This means the browser's built-in form validation will be bypassed.
- name="myForm": The name of the form is "myForm".

13. What is Box Model in CSS

A rectangle box is wrapped around every HTML element. The box model is used to determine the height and width of the rectangular box. The CSS Box consists of Width and height (or in the absence of that, default values and the content inside), padding, borders, margin.



- Content: Actual Content of the box where the text or image is placed.

- Padding: Area surrounding the content (Space between the border and content).
- Border: Area surrounding the padding.
- Margin: Area surrounding the border.

14. What are the different types of Selectors in CSS?

A CSS selector is the part of a CSS ruleset that actually selects the content you want to style. Different types of selectors are listed below.

Universal Selector: The universal selector works like a wildcard character, selecting all elements on a page. In the given example, the provided styles will get applied to all the elements on the page.

```
* {
  color: "green";
  font-size: 20px;
  line-height: 25px;
}
```

Element Type Selector: This selector matches one or more HTML elements of the same name. In the given example, the provided styles will get applied to all the ul elements on the page.

```
ul {
  line-style: none;
  border: solid 1px #ccc;
}
```

ID Selector: This selector matches any HTML element that has an ID attribute with the same value as that of the selector. In the given example, the provided styles will get applied to all the elements having ID as a container on the page.

```
#container {
  width: 960px;
  margin: 0 auto;
}

<div id="container"></div>
```

Class Selector: The class selector also matches all elements on the page that have their class attribute set to the same value as the class. In the given example, the provided styles will get applied to all the elements having ID as the box on the page.

```
.box {
  padding: 10px;
  margin: 10px;
  width: 240px;
}
<div class="box"></div>
```

Descendant Combinator: The descendant selector or, more accurately, the descendant combinator lets you combine two or more selectors so you can be more specific in your selection method.

```
#container .box {
        float: left;
        padding-bottom: 15px;
}

<div id="container">
        <div class="box"></div>
        <div class="box-2"></div>
</div>
<div class=”box”></div>
```

This declaration block will apply to all elements that have a class of box that is inside an element with an ID of the container. It’s worth noting that the .box element doesn’t have to be an immediate child: there could be another element wrapping the .box, and the styles would still apply.

Child Combinator: A selector that uses the child combinator is similar to a selector that uses a descendant combinator, except it only targets immediate child elements.

```
#container > .box {
    float: left;
    padding-bottom: 15px;
}
<div id="container">
    <div class="box"></div>
    <div>
        <div class="box"></div>
    </div>
</div>
```

The selector will match all elements that have a class of box and that are immediate children of the #container element. That means, unlike the descendant combinator, there can’t be another element wrapping the .box; it has to be a direct child element.

General Sibling Combinator: A selector that uses a general sibling combinator to match elements based on sibling relationships. The selected elements are beside each other in the HTML.

```
h2 ~ p {
    margin-bottom: 20px;
}

<h2>Title</h2>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
<div class=”box”>
    <p>Paragraph example.</p>
</div>
```

In this example, all paragraph elements (<p>) will be styled with the specified rules, but only if they are siblings of <h2> elements. There could be other elements in between the <h2> and <p>, and the styles would still apply.

Adjacent Sibling Combinator: A selector that uses the adjacent sibling combinator uses the plus symbol (+), and is almost the same as the general sibling selector. The difference is that the targeted element must be an immediate sibling, not just a general sibling.

```
p + p {
        text-indent: 1.Sem;
        margin-bottom: 0;
}


<h2>Title</h2>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
<p>Paragraph example.</p>
<div class=”box”>
        <p>Paragraph example.</p>
        <p>Paragraph example.</p>
</div>
```

The above example will apply the specified styles only to paragraph elements that immediately follow other paragraph elements. This means the first paragraph element on a page would not receive these styles. Also, if another element appeared between two paragraphs, the second paragraph of the two wouldn’t have the styles applied.

Attribute Selector: The attribute selector targets elements based on the presence and/or value of HTML attributes, and is declared using square brackets.

```
input [type=”text”] {
        background-color: #444;
        width: 200px;
}
```

```
<input type="text">
```

15. What are Pseudo elements and Pseudo classes?

Pseudo-elements allows us to create items that do not normally exist in the document tree, for example ::after.

- ::before
- ::after
- ::first-letter
- ::first-line
- ::selection

In the below example, the color will appear only on the first line of the paragraph.

```
p: :first-line {
        color: #ffOOOO;
        font-variant: small-caps;
}
```

Pseudo-classes select regular elements but under certain conditions like when the user is hovering over the link.

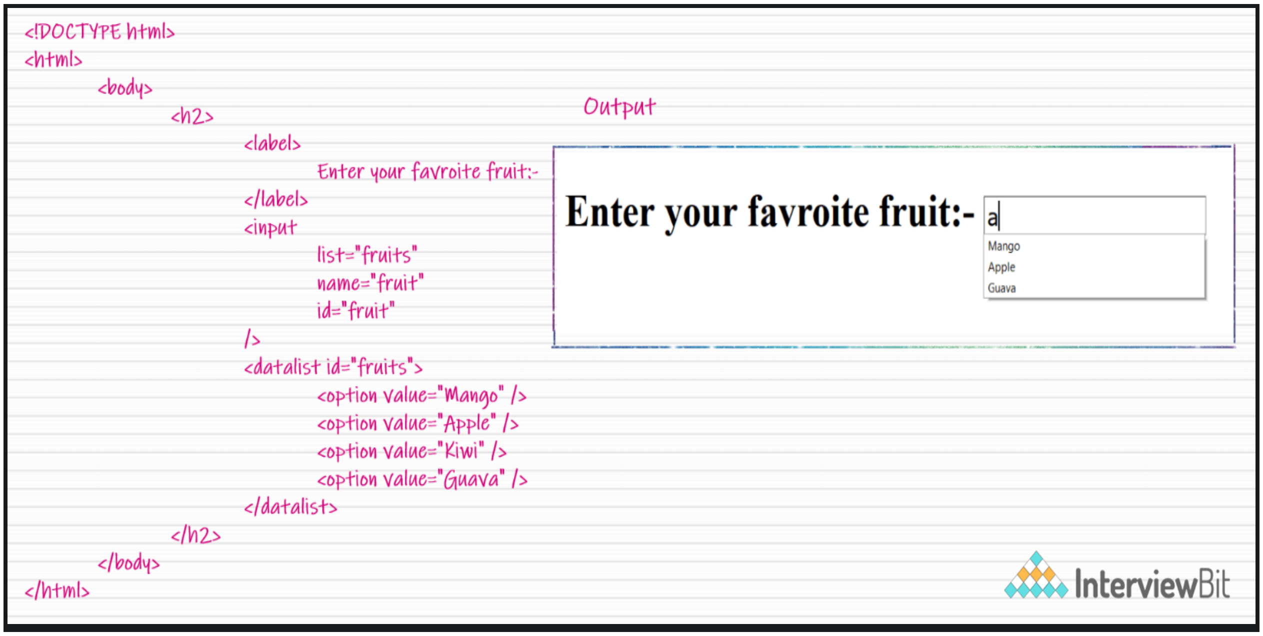
- :link
- :visited
- :hover
- :active
- :focus

Example of the pseudo-class, In the below example, the color applies to the anchor tag when it's hovered.

```
/* mouse over link */
a:hover {
        color: #FFOOFF;
}
```

16. Is the <datalist> tag and <select> tag same?

No. The <datalist> tag and <select> tag are different. In the case of <select> tag a user will have to choose from a list of options, whereas <datalist> when used along with the <input> tag provides a suggestion that the user selects one of the options given or can enter some entirely different value.



```
<!DOCTYPE html>
<html>
        <body>
                <h2>
                        <label>
                                Enter your favroite fruit:-
                        </label>
                        <input
                                list="fruits"
                                name="fruit"
                                id="fruit"
                        />
                        <datalist id="fruits">
                                <option value="Mango" />
                                <option value="Apple" />
                                <option value="Kiwi" />
                                <option value="Guava" />
                        </datalist>
                </h2>
        </body>
</html>
```

Output

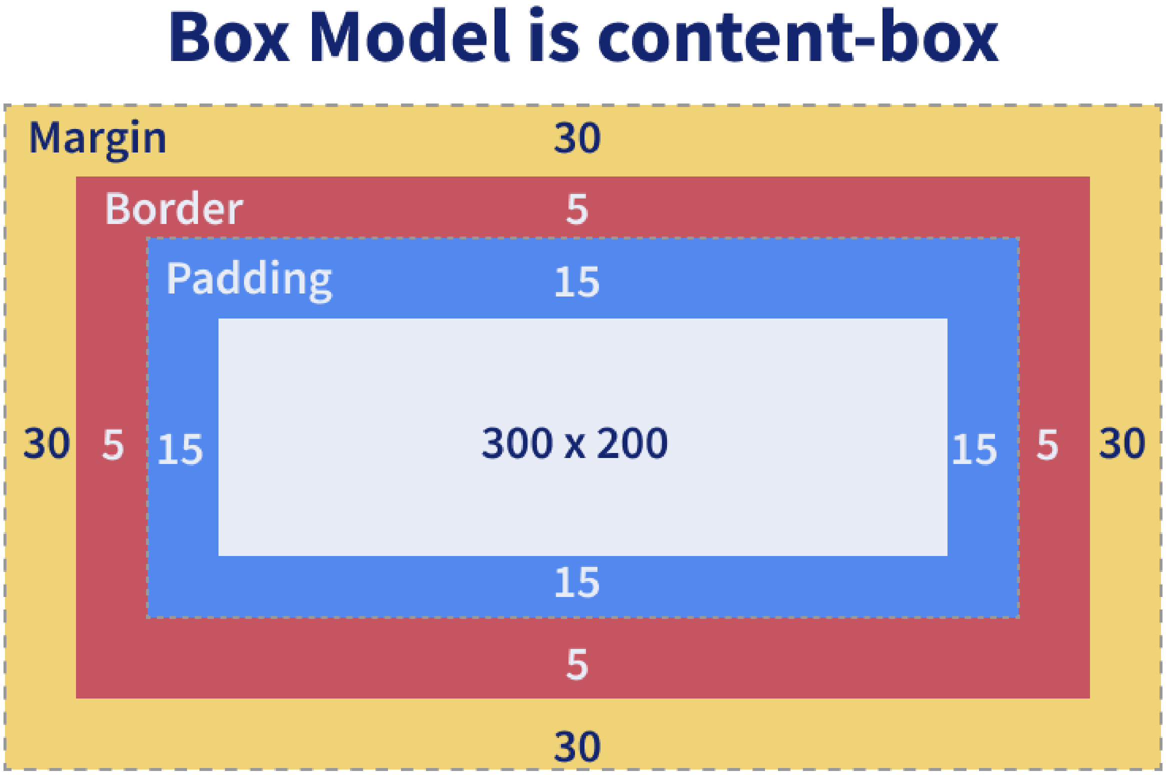
```
Enter your favroite fruit:- a
                            Mango
                            Apple
                            Guava
```

17. How is border-box different from content-box?

The **content-box** is the default value box-sizing property. The height and the width properties consist only of the content by excluding the border and padding. Consider an example as shown:

```
div{
   width:300px;
   height:200px;
   padding:15px;
   border: 5px solid grey;
   margin:30px;
   box-sizing:content-box;
}
```

Here, the box-sizing for the div element is given as content-box. That means, the height and width considered for the div content exclude the padding and border. We will get full height and width parameters specified for the content as shown in the below image.



InterviewBit

The **border-box** property includes the content, padding and border in the height and width properties. Consider an example as shown:

```
div{
    width:300px;
    height:200px;
    padding:15px;
    border: 5px solid grey;
    margin:30px;
   box-sizing:border-box;
}
```

Here, the box-sizing for the div element is given as border-box. That means the height and width considered for the div content will also include the padding and border. This means that the actual height of the div content will be:

actual height = height -

padding on top and bottom -

border on top and bottom

= 200 - (15*2) - (5*2)

= 160 px

and the actual width of the div content would be:

actual width = width -

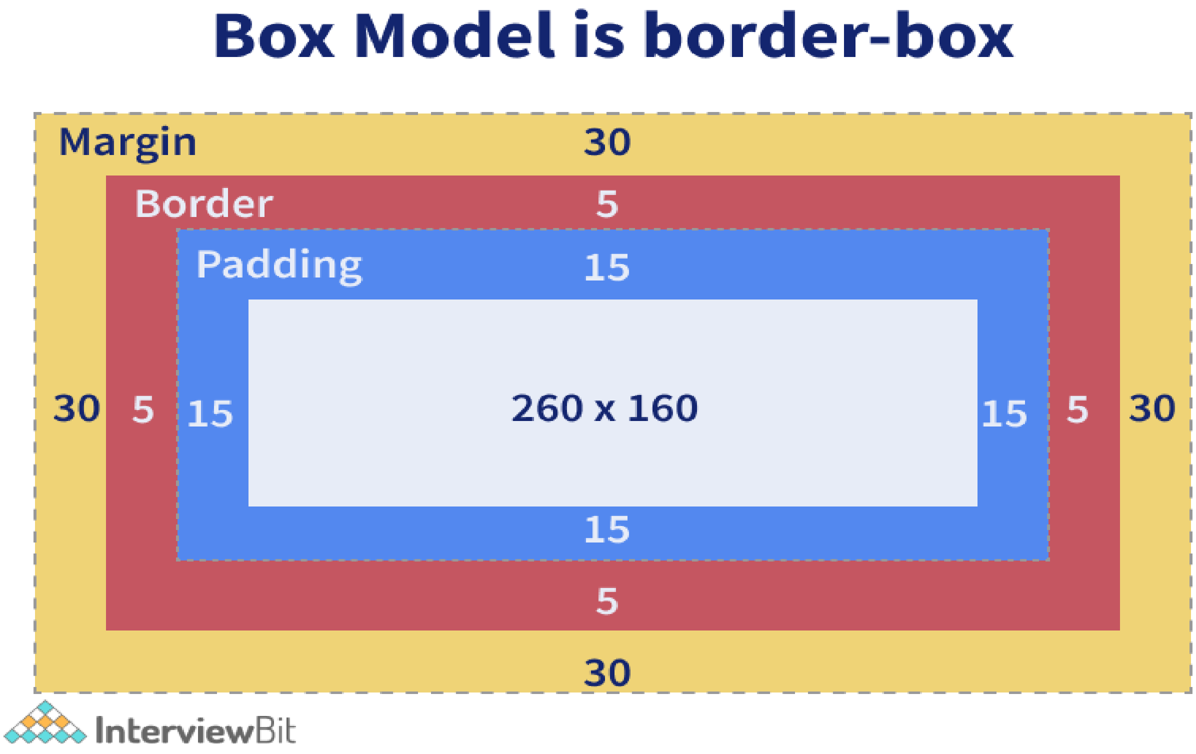
padding on right and left -

border on right and left

= 300 - (15*2) - (5*2)

= 260 px

This is represented in the image below:



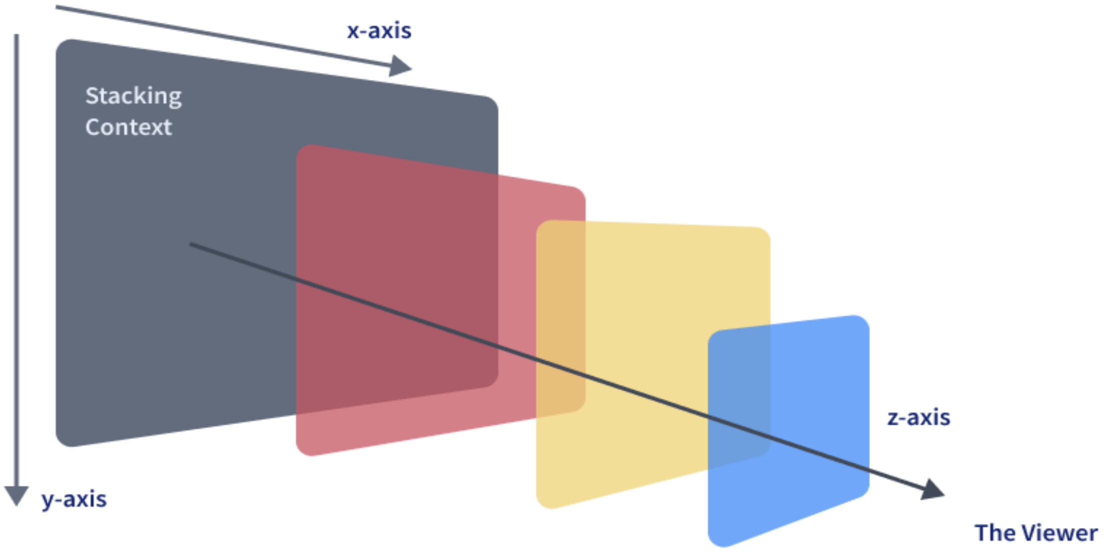
18. What is a z-index, how does it function?

z-index is used for specifying the vertical stacking of the overlapping elements that occur at the time of its positioning (it works with position absolute,relative,sticky and fixed only). It specifies the vertical stack order of the elements positioned that helps to define how the display of elements should happen in cases of overlapping.

The default value of this property is 0 and can be either positive or negative. Apart from 0, the values of the z-index can be:

- Auto: The stack order will be set equal to the parent.
- Number: The number can be positive or negative. It defines the stack order.
- Initial: The default value of 0 is set to the property.
- Inherit: The properties are inherited from the parent.

The elements having a lesser value of z-index are stacked lower than the ones with a higher z-index.



The z-index property determines the order of the element on the z-axis of the stacking contect.

InterviewBit

From the above figure, we can see that as the value of the z-index increases along the z-axis, the order of stacking would be towards the top of other elements along the vertical axis.

19. What is VH/VW (viewport height/ viewport width) in CSS?

It’s a CSS unit used to measure the height and width in percentage with respect to the viewport. It is used mainly in responsive design techniques. The measure VH is equal to 1/100 of the height of the viewport. If the height of the browser is 1000px, 1vh is equal to 10px. Similarly, if the width is 1000px, then 1 vw is equal to 10px

20. What are the properties of flexbox?

Flexbox stands for flexible box and it was introduced around 2017 in CSS with the purpose of providing an efficient way to handle layouts, align elements within them and distribute spaces amongst the items in dynamic/responsive conditions. It provides an enhanced ability to alter the dimensions of the items and make use of the available space in the container efficiently. In order to achieve this, CSS3 provides some properties.

The properties of flexbox are as follows:

- flex-direction: This property helps in defining the direction the container should stack the items targeted for flex. The values of this property can be
    - row: Stacks items horizontally from left to right in the flex container.
    - column: Stacks items vertically from top to bottom in the flex container.
    - row-reverse: Stacks items horizontally from right to left in the flex container.
    - column-reverse: Stacks items vertically from bottom to top in the flex container.
- flex-wrap: This property specifies of the flex items should be wrapped or not. Possible values are:
    - wrap: The flex items would be wrapped if needed.
    - nowrap: This is the default value that says the items won’t be wrapped.
    - wrap-reverse: This specifies that the items will be wrapped if needed but in reverse order.
- flex-flow: This property is used for setting both flex-direction and flex-wrap properties in one statement.
- justify-content: Used for aligning the flex items. Possible values are:
    - center: It means that all the flex items are present at the center of the container.
    - flex-start: This value states that the items are aligned at the start of the container. This is the default value.
    - flex-end: This value ensures the items are aligned at the end of the container.

  - space-around: This value displays the items having space between, before, around the items.
  - space-between: This value displays items with spaces between the lines.
- align-items: This is used for aligning flex items

RESOURCES TO LEARN CSS

- https://web.dev/learn/css/
- https://www.codecademy.com/learn/learn-css
- https://developer.mozilla.org/en-US/docs/Learn/CSS
- https://www.javatpoint.com/css-tutorial

JAVASCRIPT

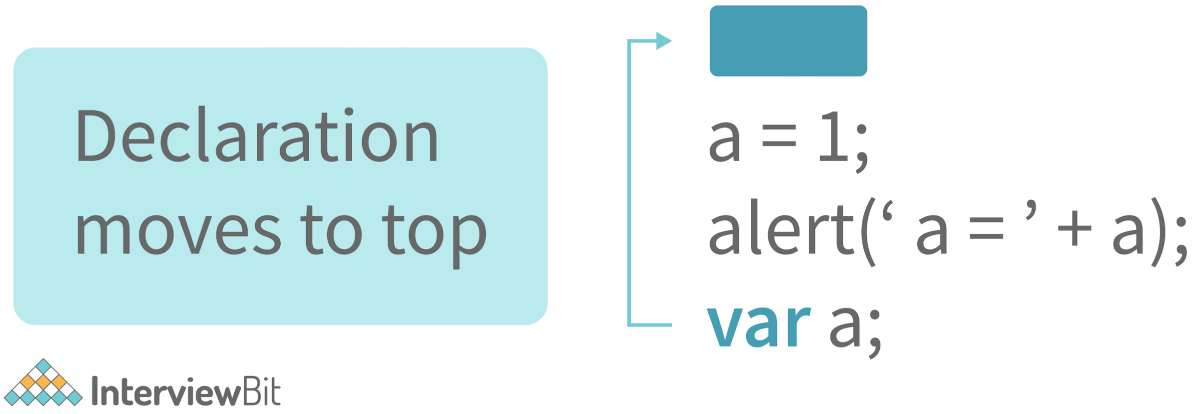
1. What are the various data types that exist in JavaScript?

These are the different types of data that JavaScript supports:

- Boolean - For true and false values
- Null - For empty or unknown values
- Undefined - For variables that are only declared and not defined or initialized
- Number - For integer and floating-point numbers
- String - For characters and alphanumeric values
- Object - For collections or complex values
- Symbols - For unique identifiers for objects

2. Explain Hoisting in javascript.

Hoisting is the default behavior of javascript where all the variable and function declarations are moved on top.



Declaration moves to top

```
a = 1;
alert(‘ a = ’ + a);
var a;
```

This means that irrespective of where the variables and functions are declared, they are moved on top of the scope. The scope can be both local and global.

Example 1:

```
hoistedVariable = 3;
console.log(hoistedVariable); // outputs 3 even when the variable is declared after it is initialized
var hoistedVariable;
```

Example 2:

```
hoistedFunction();  // Outputs " Hello world! " even when the function is declared after calling


function hoistedFunction(){
  console.log(" Hello world! ");
}
```

Example 3:

```
// Hoisting takes place in the local scope as well
function doSomething(){
  x = 33;
  console.log(x);
  var x;
}
```

doSomething(); // Outputs 33 since the local variable “x” is hoisted inside the local scope

Note - Variable initializations are not hoisted, only variable declarations are hoisted:

var x;

console.log(x); // Outputs "undefined" since the initialization of "x" is not hoisted

x = 23;

Note - To avoid hoisting, you can run javascript in strict mode by using “use strict” on top of the code:

"use strict";

x = 23; // Gives an error since 'x' is not declared

var x;

3. Difference between “ == “ and “ === “ operators.

Both are comparison operators. The difference between both the operators is that “==” is used to compare values whereas, “ === “ is used to compare both values and types.

Example:

var x = 2;

var y = "2";

(x == y) // Returns true since the value of both x and y is the same

(x === y) // Returns false since the typeof x is "number" and typeof y is "string"

4. Explain Implicit Type Coercion in javascript.

Implicit type coercion in javascript is the automatic conversion of value from one data type to another. It takes place when the operands of an expression are of different data types.

Example : String coercion takes place while using the ‘ + ‘ operator. When a number is added to a string, the number type is always converted to the string type.

Example 1:

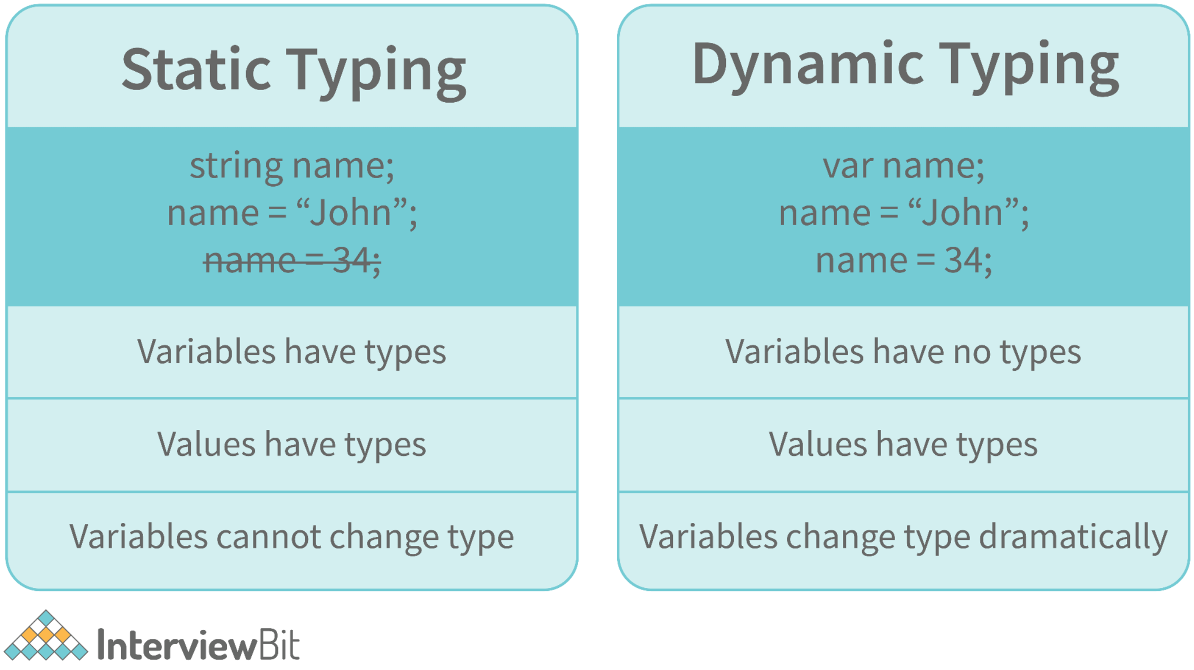
var x = 3;

var y = "3";

x + y // Returns "33"

5. Is javascript a statically typed or a dynamically typed language?

JavaScript is a dynamically typed language. In a dynamically typed language, the type of a variable is checked during run-time in contrast to a statically typed language, where the type of a variable is checked during compile-time.



Static Typing	Dynamic Typing

string name;
name = “John”;
~~name = 34;~~
	var name;
	name = “John”;
	name = 34;

Variables have types	Variables have no types
Values have types	Values have types
Variables cannot change type	Variables change type dramatically

InterviewBit

Since javascript is a loosely(dynamically) typed language, variables in JS are not associated with any type. A variable can hold the value of any data type.

6. What data types are passed by value and what are passed by reference.

In JavaScript, primitive data types are passed by value and non-primitive data types are passed by reference.

7. What is an Immediately Invoked Function in JavaScript?

An Immediately Invoked Function (known as IIFE and pronounced as IIFY) is a function that runs as soon as it is defined.

Syntax of IIFE :

```
(function(){
  // Do something;
})();
```

8. Explain Higher Order Functions in javascript.

Functions that operate on other functions, either by taking them as arguments or by returning them, are called higher-order functions.

Examples of higher-order functions:

```
function higherOrder(fn) {
  fn();
}
higherOrder(function() { console.log("Hello world") });

function higherOrder2() {
  return function() {
    return "Do something";
  }
}
var x = higherOrder2();
x()   // Returns "Do something"
```

9. Explain call(), apply() and, bind() methods.

1. call():

- It's a predefined method in javascript.
- This method invokes a method (function) by specifying the owner object.
- Example 1:

```
function sayHello(){
  return "Hello " + this.name;
}

var obj = {name: "Sandy"};

sayHello.call(obj);

// Returns "Hello Sandy"
```

- call() method allows an object to use the method (function) of another object.
- Example 2:

```
var person = {
  age: 23,
  getAge: function(){
    return this.age;
  }
}
var person2 = {age:  54};
person.getAge.call(person2);
// Returns 54
```

- call() accepts arguments:

```
function saySomething(message){
  return this.name + " is " + message;
}
var person4 = {name:  "John"};
saySomething.call(person4, "awesome");
// Returns "John is awesome"
```

apply()

The apply method is similar to the call() method. The only difference is that, call() method takes arguments separately whereas, apply() method takes arguments as an array.

```
function saySomething(message){
 return this.name + " is " + message;
}
var person4 = {name:  "John"};
saySomething.apply(person4, ["awesome"]);
```

bind():

- This method returns a new function, where the value of “this” keyword will be bound to the owner object, which is provided as a parameter.
- Example with arguments:

```
var bikeDetails = {
   displayDetails: function(registrationNumber,brandName){
   return this.name+ " , "+ "bike details: "+ registrationNumber + " , " + brandName;
 }
}
var person1 = {name:  "Vivek"};
var detailsOfPerson1 = bikeDetails.displayDetails.bind(person1, "TS0122", "Bullet");
// Binds the displayDetails function to the person1 object
detailsOfPerson1();
//Returns Vivek, bike details: TS0122, Bullet
```

10. What is currying in JavaScript?

Currying is an advanced technique to transform a function of arguments n, to n functions of one or fewer arguments.

Example of a curried function:

```
function add (a) {
 return function(b){
   return a + b;
 }
}
add(3)(4)
```

For Example, if we have a function f(a,b), then the function after currying, will be transformed to f(a)(b).

11. What is callback function in JS

In JavaScript, you can also pass a function as an argument to a function. This function that is passed as an argument inside of another function is called a callback function. For example,

```
// function
function greet(name, callback) {
    console.log('Hi' + ' ' + name);
    callback();
}

// callback function
function callMe() {
    console.log('I am callback function');
}

// passing function as an argument
greet('Peter', callMe);
```

Output

```
Hi Peter
I am callback function
```

In the above program, there are two functions. While calling the greet() function, two arguments (a string value and a function) are passed.

The callMe() function is a callback function.

Benefits of callback function : The benefit of using a callback function is that you can wait for the result of a previous function call and then execute another function call.

NODE.JS

1. What is Node.js?

Node.js is a virtual machine that uses JavaScript as its scripting language and runs Chrome's V8 JavaScript engine.

Node.js utilizes a non-blocking I/O model that results in swift response times and concurrent processing. Unlike other frameworks that require complicated thread management, Node.js simplifies development by enabling multiple tasks to be handled simultaneously in a single-threaded environment. This approach boosts efficiency and application performance, making development easier and more straightforward.

2. What do you understand by callback hell?

```
async_A(function(){
  async_B(function(){
    async_C(function(){
      async_D(function(){
      ....
      });
    });
  });
});
```

For the above example, we are passing callback functions and it makes the code unreadable and not maintainable, thus we should change the async logic to avoid this.

3. What are the advantages of using promises instead of callbacks?

The main advantage of using promise is you get an object to decide the action that needs to be taken after the async task completes. This gives more manageable code and avoids callback hell

4. Why is Node.js single-threaded?

Node.js was created explicitly as an experiment in async processing. This was to try a new theory of doing async processing on a single thread over the existing thread-based implementation of scaling via different frameworks.

5. If Node.js is single threaded then how does it handle concurrency?

The main loop is single-threaded and all async calls are managed by libuv library.