INTERVIEW QUESTIONS ON JAVASCRIPT

1. What are the different data types present in javascript?

To know the type of a JavaScript variable, we can use the **typeof** operator.
Primitive types

- **String** - It represents a series of characters and is written with quotes. A string can be represented using a single or a double quote.
  Example :

```
var str = "Vivek Singh Bisht"; //using double quotes
var str2 = 'John Doe'; //using single quotes
```

- **Number** - It represents a number and can be written with or without decimals.
  Example :

```
var x = 3; //without decimal
var y = 3.6; //with decimal
```

- **BigInt** - This data type is used to store numbers which are above the limitation of the Number data type. It can store large integers and is represented by adding "n" to an integer literal.
  Example :

```
var bigInteger =  234567890123456789012345678901234567890;
```

- **Boolean** - It represents a logical entity and can have only two values : true or false. Booleans are generally used for conditional testing.
  Example :

```
var a = 2;
var b =  3;
var c =  2;
(a == b) // returns false
(a == c) //returns true
```

- **Undefined** - When a variable is declared but not assigned, it has the value of undefined and it's type is also undefined.
  Example :

```
var x; // value of x is undefined
```

```
var y = undefined; // we can also set the value of a variable as undefined
```

- **Null -** It represents a non-existent or a invalid value.
  Example :

```
var z = null;
```

- **Symbol** - It is a new data type introduced in the ES6 version of javascript. It is used to store an anonymous and unique value.
  Example :

```
var symbol1 = Symbol('symbol');
```

- typeof **of primitive types** :

```
typeof "John Doe" // Returns "string"
typeof 3.14 // Returns "number"
typeof true // Returns "boolean"
typeof 2345678901234567890123456789012345678901234567890n // Returns bigint
typeof undefined // Returns "undefined"
typeof null // Returns "object" (kind of a bug in JavaScript)
typeof Symbol('symbol') // Returns Symbol
```

Non-primitive types

Primitive data types can store only a single value. To store multiple and complex values, non-primitive data types are used.

Object - Used to store collection of data.

Example:

```
// Collection of data in key-value pairs

var obj1 = {
  x:  43,
  y:  "Hello world!",
  z: function(){
```

```
    return this.x;
  }
}

// Collection of data as an ordered list

var array1 = [5, "Hello", true, 4.1];
```

***Note- It is important to remember that any data type that is not primitive data type, is of Object type in javascript.**

2. Explain Hoisting in javascript.

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

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.

****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 value 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 automatic conversion of value from one data type to another. It takes place when the operands of an expression are of different data types.

String coercion

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"
```

Example 2:

```
var x = 24;
var y = "Hello";
x + y   // Returns "24Hello";
```

****Note - ‘ + ‘ operator when used to add two numbers, outputs a number. The same ‘ + ‘ operator when used to add two strings, outputs the concatenated string:**

```
var name = "Vivek";
var surname = " Bisht";

name + surname     // Returns "Vivek Bisht"
```

Let’s understand both the examples where we have added a number to a string,

When JavaScript sees that the operands of the expression x + y are of different types (one being a number type and the other being a string type) , it converts the number type to the string type and then performs the operation. Since after conversion, both the variables are of string type, the ‘ + ‘ operator outputs the concatenated string “33”

in the first example and “24Hello” in the second example.

****Note - Type coercion also takes place when using the ‘ - ‘ operator, but the difference while using ‘ - ‘ operator is that, a string is converted to a number and then subtraction takes place.**

```
var x = 3;
Var y = "3";
x - y    //Returns 0 since the variable y (string type) is converted to a number type
```

Boolean Coercion

Boolean coercion takes place when using logical operators, ternary operators, if statements and loop checks. To understand boolean coercion in if statements and operators, we need to understand truthy and falsy values.

Truthy values are those which will be converted (coerced) to **true** . Falsy values are those which will be converted to **false** .

All values except **false, 0, 0n, -0, “”, null, undefined and NaN** are truthy values.

If statements:

Example:

```
var x = 0;
var y = 23;

if(x) { console.log(x) }   // The code inside this block will not run since the value of x i
s 0(Falsy)

if(y) { console.log(y) }    // The code inside this block will run since the value of y is 2
3 (Truthy)
```

Logical operators:

Logical operators in javascript, unlike operators in other programming languages, **do not return true or false. They always return one of the operands.**

OR (| |) operator - If the first value is truthy, then the first value is returned. Otherwise, always the second value gets returned.

AND (&&) operator - If both the values are truthy, always the second value is returned. If the first value is falsy then the first value is returned or if the second value is falsy then the second value is returned.

Example:

```
var x = 220;
var y = "Hello";
var z = undefined;

x | | y    // Returns 220 since the first value is truthy

x | | z   // Returns 220 since the first value is truthy

x && y    // Returns "Hello" since both the values are truthy

y && z   // Returns undefined since the second value is falsy

if( x && y ){
  console.log("Code runs" ); // This block runs because x && y returns "Hello" (Truthy )
}

if( x || z ){
  console.log("Code runs");  // This block runs because x || y returns 220(Truthy)
}
```

Equality Coercion

Equality coercion takes place when using ‘ == ‘ operator. As we have stated before

The ‘ == ‘ operator compares values and not types.

While the above statement is a simple way to explain == operator, it’s not completely

true

The reality is that while using the '==' operator, coercion takes place.

The '==' operator, converts both the operands to the same type and then compares them.

Example:

```
var a = 12;
var b = "12";
a == b // Returns true because both 'a' and 'b' are converted to the same type and then compared. Hence the operands are equal.
```

Coercion does not take place when using the '===' operator. Both operands are not converted to the same type in the case of '===' operator.

Example:

```
var a = 226;
var b = "226";

a === b // Returns false because coercion does not take place and the  operands are of different types. Hence they are not equal.
```

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 statically typed language, where the type of a variable is checked during **compile-time.**

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.
For example, a variable which is assigned a number type can be converted to a string type:

```
var a = 23;
```

```
var a = "Hello World!";
```

6. What is NaN property in JavaScript?

NaN property represents **"Not-a-Number"** value. It indicates a value which is not a legal number.

typeof of a NaN will return a **Number** .

To check if a value is NaN, we use the **isNaN()** function,

****Note- isNaN() function converts the given value to a Number type, and then equates to NaN.**

```
isNaN("Hello")  // Returns true

isNaN(345)   // Returns false

isNaN('1')  // Returns false, since '1' is converted to Number type which results in 0 ( a number)

isNaN(true) // Returns false, since true converted to Number type results in 1 ( a numb er)

isNaN(false) // Returns false

isNaN(undefined) // Returns true
```

7. Explain passed by value and passed by reference.

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

For understanding passed by value and passed by reference, we need to understand what happens when we create a variable and assign a value to it,

```
var x = 2;
```

In the above example, we created a variable x and assigned it a value "2". In the background, the "=" (assign operator) allocates some space in the memory, stores the value "2" and returns the location of the allocated memory space. Therefore, the variable x in the above code points to the location of the memory space instead of pointing to the value 2 directly.

Assign operator behaves differently when dealing with primitive and non primitive data types,

Assign operator dealing with primitive types:

```
var y = 234;
var z = y;
```

In the above example, assign operator knows that the value assigned to y is a primitive type (number type in this case), so when the second line code executes, where the value of y is assigned to z, the assign operator takes the value of y (234) and allocates a new space in the memory and returns the address. Therefore, variable z is not pointing to the location of variable y, instead it is pointing to a new location in the memory.

```
var y = #8454; // y pointing to address of the value 234

var z = y;

var z = #5411; // z pointing to a completely new address of the value 234

// Changing the value of y
y = 23;
console.log(z);  // Returns 234, since z points to a new address in the memory so changes in y will not effect z
```

From the above example, we can see that primitive data types when passed to another variable, are passed by value. Instead of just assigning the same address to another variable, the value is passed and new space of memory is created.

Assign operator dealing with non-primitive types:

```
var obj = { name: "Vivek", surname: "Bisht" };


var obj2 = obj;
```

In the above example, the assign operator, directly passes the location of the variable obj to the variable obj2. In other words, the reference of the variable obj is passed to the variable obj2.

```
var obj = #8711;  // obj pointing to address of { name: "Vivek", surname: "Bisht" }


var obj2 = obj;


var obj2 = #8711; // obj2 pointing to the same address


// changing the value of obj1


obj1.name = "Akki";


console.log(obj2);


// Returns {name:"Akki", surname:"Bisht"} since both the variables are pointing to the same address.
```

From the above example, we can see that while passing non-primitive data types, the assign operator directly passes the address (reference).

Therefore, non-primitive data types are always **passed by reference.**

8. 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;
})();
```

To understand IIFE, we need to understand the two sets of parentheses which are added while creating an IIFE :

First set of parenthesis:

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

While executing javascript code, whenever the compiler sees the word “function”, it assumes that we are declaring a function in the code. Therefore, if we do not use the first set of parentheses, the compiler throws an error because it thinks we are declaring a function, and by the syntax of declaring a function, a function should always have a name.

```
function() {
  //Do something;
}
// Compiler gives an error since the syntax of declaring a function is wrong in the code above.
```

To remove this error, we add the first set of parenthesis that tells the compiler that the function is not a function declaration, instead, it’s a function expression.

Second set of parenthesis:

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

From the definition of an IIFE, we know that our code should run as soon as it is defined. A function runs only when it is invoked. If we do not invoke the function, the function declaration is returned:

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

// Returns the function declaration
```

Therefore to invoke the function, we use the second set of parenthesis. .

9. 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.

Higher order functions are a result of functions being **first-class citizens** in javascript.

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"
```

10. Explain “this” keyword.

The “this” keyword refers to the object that the function is a property of.

The value of “this” keyword will always depend on the object that is invoking the function.

Confused? Let’s understand the above statements by examples:

```
function doSomething() {
  console.log(this);
}

doSomething();
```

What do you think the output of the above code will be?

****Note - Observe the line where we are invoking the function.**

Check the definition again:

The “this” keyword refers to the object that the function is a property of.

In the above code, function is a property of which object?

Since the function is invoked in the global context, **the function is a property of the global object.**

Therefore, the output of the above code will be **the global object.** Since we ran the above code inside the browser, the global object is **the window object.**

Example 2:

```
var obj = {
   name:  "vivek",
   getName: function(){
   console.log(this.name);
  }
}

obj.getName();
```

In the above code, at the time of invocation, the getName function is a property of the object **obj** , therefore, the **this** keyword will refer to the object **obj** , and hence the output will be “vivek”.

Example 3:

```
 var obj = {
   name:  "vivek",
   getName: function(){
   console.log(this.name);
  }

}

var getName = obj.getName;

var obj2 = {name:"akshay", getName };
obj2.getName();
```

Can you guess the output here?

The output will be “akshay”.

Although the getName function is declared inside the object **obj** , at the time of

invocation, getName() is a property of **obj2** , therefore the “this” keyword will refer to **obj2** .

The silly way to understanding the **this** keyword is, whenever the function is invoked, check the object before the **dot** . The value of **this** . keyword will always be the object before the **dot** .

If there is no object before the dot like in example1, the value of this keyword will be the global object.

Example 4:

```
var obj1 = {
   address : "Mumbai,India",
   getAddress: function(){
   console.log(this.address);
  }
}

var getAddress = obj1.getAddress;
var obj2 = {name:"akshay"};
obj2.getAddress();
```

Can you guess the output?

The output will be an error.

Although in the code above, the this keyword refers to the object **obj2** , obj2 does not have the property “address”‘, hence the getAddress function throws an error.

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

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: TS0452, Thunderbird
```

12. What is currying in JavaScript?

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

By using the currying technique, we do not change the functionality of a function, we just change the way it is invoked.

Let's see currying in action:

```
function multiply(a,b){
  return a*b;
}

function currying(fn){
  return function(a){
    return function(b){
      return fn(a,b);
    }
  }
}

var curriedMultiply = currying(multiply);

multiply(4, 3); // Returns 12

curriedMultiply(4)(3); // Also returns 12
```

As one can see in the code above, we have transformed the function **multiply(a,b)** to a function **curriedMultiply** , which takes in one parameter at a time.

13. Explain Scope and Scope Chain in javascript.

Scope in JS, determines the accessibility of variables and functions at various parts in one's code.

In general terms, the scope will let us know at a given part of code, what are the variables and functions that we can or cannot access.

There are three types of scopes in JS:

- Global Scope
- Local or Function Scope
- Block Scope

Global Scope

Variables or functions declared in the global namespace have global scope, which means all the variables and functions having global scope can be accessed from anywhere inside the code

```
var globalVariable = "Hello world";


function sendMessage(){
  return globalVariable; // can access globalVariable since it's written in global space
}


function sendMessage2(){
  return sendMessage(); // Can access sendMessage function since it's written in globa
l space
}


sendMessage2();  // Returns “Hello world”
```

Function Scope

Any variables or functions declared inside a function have local/function scope, which means that all the variables and functions declared inside a function, can be accessed from within the function and not outside of it.

```
function awesomeFunction(){
  var a = 2;
```

```
  var multiplyBy2 = function(){
    console.log(a*2); // Can access variable "a" since a and multiplyBy2 both are written inside the same function
  }
}
console.log(a); // Throws reference error since a is written in local scope and cannot be accessed outside

multiplyBy2(); // Throws reference error since multiplyBy2 is written in local scope
```

Block Scope

Block scope is related to the variables declared using let and const. Variables declared with var do not have block scope.

Block scope tells us that any variable declared inside a block { }, can be accessed only inside that block and cannot be accessed outside of it.

```
{
  let x = 45;
}

console.log(x); // Gives reference error since x cannot be accessed outside of the block

for(let i=0; i<2; i++){
  // do something
}

console.log(i); // Gives reference error since i cannot be accessed outside of the for loop block
```

Scope Chain

JavaScript engine also uses Scope to find variables.

Let’s understand that using an example:

```
var y = 24;


function favFunction(){
  var x = 667;
  var anotherFavFunction = function(){
    console.log(x); // Does not find x inside anotherFavFunction, so looks for variable i
nside favFunction, outputs 667
  }


  var yetAnotherFavFunction = function(){
    console.log(y); // Does not find y inside yetAnotherFavFunction, so looks for variab
le inside favFunction and does not find it, so looks for variable in global scope, finds it
and outputs 24
  }


  anotherFavFunction();
  yetAnotherFavFunction();
}



favFunction();
```

As you can see in the code above, if the javascript engine does not find the variable in local scope, it tries to check for the variable in the outer scope. If the variable does not exist in the outer scope, it tries to find the variable in the global scope.

If the variable is not found in the global space as well, reference error is thrown.

14. Explain Closures in JavaScript.

Closures is an ability of a function to remember the variables and functions that are declared in its outer scope.

```
var Person = function(pName){
  var name = pName;

  this.getName = function(){
    return name;
  }
}

var person = new Person("Neelesh");
console.log(person.getName());
```

Let's understand closures by example:

```
function randomFunc(){
  var obj1 = {name:"Vivian", age:45};

  return function(){
    console.log(obj1.name + " is "+ "awesome"); // Has access to obj1 even when the ra
ndomFunc function is executed

  }
}

var initialiseClosure = randomFunc(); // Returns a function
```

```
initialiseClosure();
```

Let's understand the code above,

The function randomFunc() gets executed and returns a function when we assign it to a variable:

```
var initialiseClosure = randomFunc();
```

The returned function is then executed when we invoke initialiseClosure:

```
initialiseClosure();
```

The line of code above outputs "Vivian is awesome" and this is possible because of closure.

When the function randomFunc() runs, it sees that the returning function is using the variable obj1 inside it:

```
console.log(obj1.name + " is "+ "awesome");
```

Therefore randomFunc(), instead of destroying the value of obj1 after execution, **saves the value in the memory for further reference.** This is the reason why the returning function is able to use the variable declared in the outer scope even after the function is already executed.

This ability of a function to store a variable for further reference even after it is executed, is called Closure.

15. What are object prototypes?

All javascript objects inherit properties from a prototype.

For example,

Date objects inherit properties from the Date prototype

Math objects inherit properties from the Math prototype

Array objects inherit properties from the Array prototype.

On top of the chain is **Object.prototype.** Every prototype inherits properties and methods from the Object.prototype.

A prototype is a blueprint of an object. Prototype allows us to use properties and methods on an object even if the properties and methods do not exist on the current object.

Let's see prototypes help us use methods and properties:

```
var arr = [];
arr.push(2);


console.log(arr); // Outputs [2]
```

In the code above, as one can see, we have not defined any property or method called push on the array "arr" but the javascript engine does not throw an error.

The reason being the use of prototypes. As we discussed before, Array objects inherit properties from the Array prototype.

The javascript engine sees that the method push does not exist on the current array object and therefore, looks for the method push inside the Array prototype and it finds the method.

Whenever the property or method is not found on the current object, the javascript engine will always try to look in its prototype and if it still does not exist, it looks inside the prototype's prototype and so on.

16. What are callbacks?

A callback is a function that will be executed after another function gets executed.

In javascript, functions are treated as first-class citizens, they can be used as an argument of another function, can be returned by another function and can be used as a property of an object.

Functions that are used as an argument to another function are called callback functions.

Example:

```
function divideByHalf(sum){
  console.log(Math.floor(sum / 2));
}

function multiplyBy2(sum){
  console.log(sum * 2);
}

function operationOnSum(num1,num2,operation){
  var sum = num1 + num2;
  operation(sum);
}

operationOnSum(3, 3, divideByHalf); // Outputs 3

operationOnSum(5, 5, multiplyBy2); // Outputs 20
```

In the code above, we are performing mathematical operations on the sum of two numbers.

The operationOnSum function takes 3 arguments, first number, second number, and the operation that is to be performed on their sum (callback) .

Both divideByHalf and multiplyBy2 functions are used as callback functions in the code above.

These callback functions will be executed only after the function operationOnSum is executed.

Therefore, callback is a function that will be executed after another function gets executed.

17.What is memoization?

Memoization is a form of caching where the return value of a function is cached based on its parameters. If the parameter of that function is not changed, the cached version of the function is returned.

Let's understand memoization, by converting a simple function to a memoized function:

****Note- Memoization is used for expensive function calls but in the following example, we are considering a simple function for understanding the concept of memoization better.**

Consider the following function:

```
function addTo256(num){
  return num + 256;
}

addTo256(20); // Returns 276
addTo256(40); // Returns 296
addTo256(20); // Returns 276
```

In the code above, we have written a function that adds the parameter to 256 and returns it.

When we are calling the function addTo256 again with the same parameter ("20" in the case above), we are computing the result again for the same parameter.

Computing the result with the same parameter again and again is not a big deal in the above case, but imagine if the function does some heavy duty work, then, computing the result again and again with the same parameter will lead to wastage of time.

This is where memoization comes in, by using memoization we can store(cache) the computed results based on the parameters. If the same parameter is used again while invoking the function, instead of computing the result, we directly return the stored (cached) value.

Let’s convert the above function addTo256, to a memoized function:

```
function memoizedAddTo256(){
  var cache = {};

  return function(num){
    if(num in cache){
      console.log("cached value");
      return cache[num]

    }
    else{
      cache[num] = num + 256;
      return cache[num];
    }
  }
}

var memoizedFunc = memoizedAddTo256();

memoizedFunc(20); // Normal return
memoizedFunc(20); // Cached return
```

In the code above, if we run memoizedFunc function with the same parameter, instead of computing the result again, it returns the cached result.

****Note- Although using memoization saves time, it results in larger consumption of memory since we are storing all the computed results.**

18. What is recursion in a programming language?

Recursion is a technique to iterate over an operation by having a function call itself repeatedly until it arrives at a result.

```
function add(number) {
  if (number <= 0) {
    return 0;
  } else {
    return number + add(number - 1);
  }
}


add(3) => 3 + add(2)
        3 + 2 + add(1)
        3 + 2 + 1 + add(0)
        3 + 2 + 1 + 0 = 6
```

Example of a recursive function:

The following function calculates the sum of all the elements in an array by using recursion:

```
function computeSum(arr){
  if(arr.length === 1){
    return arr[0];
  }
  else{
    return arr.pop() + computeSum(arr);
  }
}
```

```
computeSum([7, 8, 9, 99]); // Returns 123
```

19. What is the use of a constructor function in javascript?

Constructor functions are used to create objects in javascript.

When do we use constructor functions?

If we want to create multiple objects having similar properties and methods, constructor functions are used.

****Note- Name of a constructor function should always be written in Pascal Notation: every word should start with a capital letter.**

Example:

```
function Person(name,age,gender){
  this.name = name;
  this.age = age;
  this.gender = gender;
}



var person1 = new Person("Vivek", 76, "male");
console.log(person1);

var person2 = new Person("Courtney", 34, "female");
console.log(person2);
```

In the code above, we have created a constructor function named Person.

Whenever we want to create a new object of the type Person,

We need to create it using the new keyword:

```
var person3 = new Person("Lilly", 17, "female");
```

The above line of code will create a new object of the type Person.

Constructor functions allow us to group similar objects.

20. What is DOM?

DOM stands for Document Object Model.

DOM is a programming interface for HTML and XML documents.

When the browser tries to render a HTML document, it creates an object based on the HTML document called DOM. Using this DOM, we can manipulate or change various elements inside the HTML document.

Example of how HTML code gets converted to DOM:

21. What are arrow functions?

Arrow functions were introduced in the ES6 version of javascript.

They provide us with a new and shorter syntax for declaring functions.

Arrow functions can only be used as a function expression.

Let's compare the normal function declaration and the arrow function declaration in detail:

```
// Traditional Function Expression
var add = function(a,b){
  return a + b;
}

// Arrow Function Expression
var arrowAdd = (a,b) => a + b;
```

Arrow functions are declared without the function keyword. If there is only one returning expression then we don't need to use the return keyword as well in an arrow

function as shown in the example above. Also, for functions having just one line of code, curly braces { } can be omitted.

```
// Traditional function expression
var multiplyBy2 = function(num){
  return num * 2;
}
// Arrow function expression
var arrowMultiplyBy2 = num => num * 2;
```

If the function takes in only one argument, then the parenthesis () around the parameter can be omitted as shown in the code above.

```
var obj1 = {
  valueOfThis: function(){
    return this;
  }
}
var obj2 = {
  valueOfThis: ()=>{
    return this;
  }
}

obj1.valueOfThis(); // Will return the object obj1
obj2.valueOfThis(); // Will return window/global object
```

The biggest difference between the traditional function expression and the arrow function, is the handling of the **this** keyword.

By general definition, the **this** keyword always refers to the object that is calling the function.

As you can see in the code above, **obj1.valueOfThis()** returns obj1, since **this** keyword refers to the object calling the function.

In the arrow functions, there is no binding of the **this** keyword.

The **this** keyword inside an arrow function, does not refer to the object calling it. It rather inherits its value from the parent scope which is the window object in this case.

Therefore, in the code above, **obj2.valueOfThis()** returns the window object.

22. Differences between declaring variables using var, let and const.

Before the ES6 version of javascript, only the keyword var was used to declare variables.

With the ES6 Version, keywords let and const were introduced to declare variables.

keyword	const	let	var
global scope	no	no	yes
function scope	yes	yes	yes
block scope	yes	yes	no
can be reassigned	no	yes	yes

Let's understand the differences with examples:

Let's understand the differences with examples:

```
var variable1 = 23;


let variable2 = 89;


function catchValues(){
  console.log(variable1);
  console.log(variable2);
```

```
// Both the variables can be accessed anywhere since they are declared in the global s
cope
}

window.variable1; // Returns the value 23

window.variable2; // Returns undefined
```

The variables declared with the let keyword in the global scope behave just like the variable declared with the var keyword in the global scope.

Variables declared in the global scope with var and let keywords can be accessed from anywhere in the code.

But, there is one difference !

Variables that are declared with the var keyword in the global scope are added to the window/global object. Therefore, they can be accessed using window.variableName.

Whereas, the variables declared with the let keyword are not added to the global object, therefore, trying to access such variables using window.variableName results in an error.

var vs let in functional scope

```
function varVsLetFunction(){
  let awesomeCar1 = "Audi";
  var awesomeCar2 = "Mercedes";
}

console.log(awesomeCar1); // Throws an error
console.log(awesomeCar2); // Throws an error
```

Variables declared in a functional/local scope using **var** and **let** keywords behave exactly the same, meaning , they cannot be accessed from outside of the scope.

```
{
  var variable3 = [1, 2, 3, 4];
}

console.log(variable3); // Outputs [1,2,3,4]

{
  let variable4 = [6, 55, -1, 2];
}

console.log(variable4); // Throws error

for(let i = 0; i < 2; i++){
  //Do something
}

console.log(i); // Throws error

for(var j = 0; j < 2; i++){
  // Do something
}

console.log(j) // Outputs 2
```

In javascript, a block means the code written inside the curly braces **{}** .

Variables declared with **var** keyword do not have block scope. It means a variable declared in block scope **{}** with the **var** keyword is the same as declaring the variable in the global scope.

Variables declared with **let** keyword inside the block scope cannot be accessed from outside of the block.

Const keyword

Variables with the **const** keyword behave exactly like a variable declared with the let keyword with only one difference, **any variable declared with the const keyword cannot be reassigned.**

Example:

```
const x = {name:"Vivek"};

x = {address: "India"}; // Throws an error

x.name = "Nikhil"; // No error is thrown

const y = 23;

y = 44; // Throws an error
```

In the code above, although we can change the value of a property inside the variable declared with **const** keyword, we cannot completely reassign the variable itself.

23. What is the rest parameter and spread operator?

Both rest parameter and spread operator were introduced in the ES6 version of javascript.

Rest parameter (…)

It provides an improved way of handling parameters of a function.

Using the rest parameter syntax, we can create functions that can take a variable number of arguments.

Any number of arguments will be converted into an array using the rest parameter.

It also helps in extracting all or some parts of the arguments.

Rest parameter can be used by applying three dots (...) before the parameters.

```
function extractingArgs(...args){
  return args[1];
}

// extractingArgs(8,9,1); // Returns 9

function addAllArgs(...args){
  let sumOfArgs = 0;
  let i = 0;
  while(i < args.length){
    sumOfArgs += args[i];
    i++;
  }
  return sumOfArgs;
}

addAllArgs(6, 5, 7, 99); // Returns 117
addAllArgs(1, 3, 4); // Returns 8
```

****Note- Rest parameter should always be used at the last parameter of a function:**

```
// Incorrect way to use rest parameter
function randomFunc(a,...args,c){
//Do something
}

// Correct way to use rest parameter
function randomFunc2(a,b,...args){
//Do something
}
```

Spread operator (…)

Although the syntax of spread operator is exactly the same as the rest parameter, spread operator is used to spread an array, and object literals. We also use spread operators where one or more arguments are expected in a function call.

```
function addFourNumbers(num1,num2,num3,num4){
  return num1 + num2 + num3 + num4;
}

let fourNumbers = [5, 6, 7, 8];



addFourNumbers(...fourNumbers);
// Spreads [5,6,7,8] as 5,6,7,8

let array1 = [3, 4, 5, 6];
let clonedArray1 = [...array1];
// Spreads the array into 3,4,5,6
console.log(clonedArray1); // Outputs [3,4,5,6]



let obj1 = {x:'Hello', y:'Bye'};
let clonedObj1 = {...obj1}; // Spreads and clones obj1
console.log(obj1);

let obj2 = {z:'Yes', a:'No'};
let mergedObj = {...obj1, ...obj2}; // Spreads both the objects and merges it
console.log(mergedObj);
// Outputs {x:'Hello', y:'Bye',z:'Yes',a:'No'};
```

*****Note- Key differences between rest parameter and spread operator:**

- **Rest parameter is used to take a variable number of arguments and turns into an array while the spread operator takes an array or an object and spreads it**
- **Rest parameter is used in function declaration whereas the spread operator is used in function calls.**

24. What is the use of promises in javascript?

Promises are used to handle asynchronous operations in javascript.

Before promises, callbacks were used to handle asynchronous operations. But due to limited functionality of callback, using multiple callbacks to handle asynchronous code can lead to unmanageable code.

Promise object has four states -

- Pending - Initial state of promise. This state represents that the promise has neither been fulfilled nor been rejected, it is in the pending state.
- Fulfilled - This state represents that the promise has been fulfilled, meaning the async operation is completed.
- Rejected - This state represents that the promise has been rejected for some reason, meaning the async operation has failed.
- Settled - This state represents that the promise has been either rejected or fulfilled.

A promise is created using the **Promise** constructor which takes in a callback function with two parameters, **resolve** and **reject** respectively.

resolve is a function that will be called, when the async operation has been successfully completed.

reject is a function that will be called, when the async operation fails or if some error occurs.

Example of a promise:

Promises are used to handle asynchronous operations like server requests, for the ease of understanding, we are using an operation to calculate the sum of

three elements.

In the function below, we are returning a promise inside a function:

```
function sumOfThreeElements(...elements){
  return new Promise((resolve,reject)=>{
    if(elements.length > 3 ){
      reject("Only three elements or less are allowed");
    }
    else{
      let sum = 0;
      let i = 0;
      while(i < elements.length){
        sum += elements[i];
        i++;
      }
      resolve("Sum has been calculated: "+sum);
    }
  })
}
```

In the code above, we are calculating the sum of three elements, if the length of elements array is more than 3, promise is rejected, else the promise is resolved and the sum is returned.

We can consume any promise by attaching then() and catch() methods to the consumer.

then() method is used to access the result when the promise is fulfilled.

catch() method is used to access the result/error when the promise is rejected.

In the code below, we are consuming the promise:

```
sumOfThreeElements(4, 5, 6)
.then(result=> console.log(result))
.catch(error=> console.log(error));
// In the code above, the promise is fulfilled so the then() method gets executed

sumOfThreeElements(7, 0, 33, 41)
.then(result => console.log(result))
.catch(error=> console.log(error));
// In the code above, the promise is rejected hence the catch() method gets executed
```

25.What are classes in javascript?

Introduced in the ES6 version, classes are nothing but syntactic sugars for constructor functions.

They provide a new way of declaring constructor functions in javascript.

Below are the examples of how classes are declared and used:

```
// Before ES6 version, using constructor functions
function Student(name,rollNumber,grade,section){
  this.name = name;
  this.rollNumber = rollNumber;
  this.grade = grade;
  this.section = section;
}

// Way to add methods to a constructor function
Student.prototype.getDetails = function(){
  return 'Name: ${this.name}, Roll no: ${this.rollNumber}, Grade: ${this.grade}, Sec
tion:${this.section}';
}
```

```
let student1 = new Student("Vivek", 354, "6th", "A");
student1.getDetails();
// Returns Name: Vivek, Roll no:354, Grade: 6th, Section:A

// ES6 version classes
class Student{
  constructor(name,rollNumber,grade,section){
    this.name = name;
    this.rollNumber = rollNumber;
    this.grade = grade;
    this.section = section;
  }

  // Methods can be directly added inside the class
  getDetails(){
    return 'Name: ${this.name}, Roll no: ${this.rollNumber}, Grade:${this.grade}, Sec
tion:${this.section}';
  }
}

let student2 = new Student("Garry", 673, "7th", "C");
student2.getDetails();
// Returns Name: Garry, Roll no:673, Grade: 7th, Section:C
```

Key points to remember about classes:

- Unlike functions, classes are not hoisted. A class cannot be used before it is declared.
- A class can inherit properties and methods from other classes by using the extend keyword.

- All the syntaxes inside the class must follow the strict mode('use strict') of javascript. Error will be thrown if the strict mode rules are not followed.

26. What are generator functions?

Introduced in ES6 version, generator functions are a special class of functions.

They can be stopped midway and then continue from where it had stopped.

Generator functions are declared with the **function*** keyword instead of the normal **function** keyword:

```
function* genFunc(){
  // Perform operation
}
```

In normal functions, we use the **return** keyword to return a value and as soon as the return statement gets executed, the function execution stops:

```
function normalFunc(){
  return 22;
  console.log(2); // This line of code does not get executed
}
```

In the case of generator functions, when called, they do not execute the code, instead they return a **generator object** . This generator object handles the execution.

```
function* genFunc(){
  yield 3;
  yield 4;
}
genFunc(); // Returns Object [Generator] {}
```

The generator object consists of a method called **next()** , this method when called, executes the code until the nearest **yield** statement, and returns the yield value.

For example if we run the next() method on the above code:

```
genFunc().next(); // Returns {value: 3, done:false}
```

As one can see the next method returns an object consisting
of **value** and **done** properties.
Value property represents the yielded value.
Done property tells us whether the function code is finished or not. (Returns true if finished)
Generator functions are used to return iterators. Let's see an example where an iterator is returned:

```
function* iteratorFunc() {
  let count = 0;
  for (let i = 0; i < 2; i++) {
     count++;
     yield i;
  }
  return count;
}

let iterator = iteratorFunc();
console.log(iterator.next()); // {value:0,done:false}
console.log(iterator.next()); // {value:1,done:false}
console.log(iterator.next()); // {value:2,done:true}
```

As you can see in the code above, the last line returns **done:true** , since the code reaches the return statement.

27. Explain WeakSet in javascript.

In javascript, Set is a collection of unique and ordered elements.

Just like Set, WeakSet is also a collection of unique and ordered elements with some

key differences:

- Weakset contains only objects and no other type.
- An object inside the weakset is referenced weakly. This means, if the object inside the weakset does not have a reference, it will be garbage collected.
- Unlike Set, WeakSet only has three methods, **add()** , **delete()** and **has()** .

```
const newSet = new Set([4, 5, 6, 7]);
console.log(newSet);// Outputs Set {4,5,6,7}

const newSet2 = new WeakSet([3, 4, 5]); //Throws an error

let obj1 = {message:"Hello world"};
const newSet3 = new WeakSet([obj1]);
console.log(newSet3.has(obj1)); // true
```

28. Explain WeakMap in javascript.

In javascript, Map is used to store key-value pairs. The key-value pairs can be of both primitive and non-primitive types.

WeakMap is similar to Map with key differences:

- The keys and values in weakmap should always be an object.
- If there are no references to the object, the object will be garbage collected.

```
const map1 = new Map();
map1.set('Value', 1);
```

```
const map2 = new WeakMap();
map2.set('Value', 2.3); // Throws an error

let obj = {name:"Vivek"};
const map3 = new WeakMap();
map3.set(obj, {age:23});
```

29. What is Object Destructuring?

Object destructuring is a new way to extract elements from an object or an array.

Object destructuring:

Before ES6 version:

```
const classDetails = {
  strength: 78,
  benches: 39,
  blackBoard:1
}

const classStrength = classDetails.strength;
const classBenches = classDetails.benches;
const classBlackBoard = classDetails.blackBoard;
```

The same example using object destructuring:

```
const classDetails = {
  strength: 78,
  benches: 39,
  blackBoard:1
}
```

```
const {strength:classStrength, benches:classBenches,blackBoard:classBlackBoard} = classDetails;


console.log(classStrength); // Outputs 78
console.log(classBenches); // Outputs 39
console.log(classBlackBoard); // Outputs 1
```

As one can see, using object destructuring we have extracted all the elements inside an object in one line of code.

If we want our new variable to have the same name as the property of an object we can remove the colon:

```
const {strength:strength} = classDetails;
// The above line of code can be written as:
const {strength} = classDetails;
```

Array destructuring:

Before ES6 version:

```
const arr = [1, 2, 3, 4];
const first = arr[0];
const second = arr[1];
const third = arr[2];
const fourth = arr[3];
```

The same example using object destructuring:

```
const arr = [1, 2, 3, 4];
```

```
const [first,second,third,fourth] = arr;


console.log(first); // Outputs 1
console.log(second); // Outputs 2
console.log(third); // Outputs 3
console.log(fourth); // Outputs 4
```

30. What is a Temporal Dead Zone?

Temporal Dead Zone is a behaviour that occurs with variables declared using **let** and **const** keywords.

It is a behaviour where we try to access a variable before it is initialized.

Examples of temporal dead zone:

```
x = 23; // Gives reference error


let x;



function anotherRandomFunc(){
  message = "Hello"; // Throws a reference error


  let message;
}
anotherRandomFunc();
```

In the code above, both in global scope and functional scope, we are trying to access variables which have not been declared yet. This is called the **Temporal Dead Zone** .

1) What is JavaScript?

JavaScript is *a scripting language*. It is different from Java language. It is object-based, lightweight, cross-platform translated language. It is widely used for client-side validation. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser. More details.

2) List some features of JavaScript.

Some of the features of JavaScript are:

- Lightweight
- Interpreted programming language
- Good for the applications which are network-centric
- Complementary to Java
- Complementary to HTML
- Open source

  Cross-platform

3) Who developed JavaScript, and what was the first name of JavaScript?

JavaScript was developed by Brendan Eich, who was a Netscape programmer. Brendan Eich developed this new scripting language in just ten days in the year September 1995. At the time of its launch, JavaScript was initially called Mocha. After that, it was called Live Script and later known as JavaScript.

4) List some of the advantages of JavaScript.

Some of the advantages of JavaScript are:

- Server interaction is less
- Feedback to the visitors is immediate
- Interactivity is high
- Interfaces are richer

5) List some of the disadvantages of JavaScript.

Some of the disadvantages of JavaScript are:

- No support for multithreading
- No support for multiprocessing

- Reading and writing of files is not allowed
- No support for networking applications.

6) Define a named function in JavaScript.

The function which has named at the time of definition is called a named function. For example

```
1. function msg()
2. {
3.   document.writeln("Named Function");
4. }
5. msg();
```

7) Name the types of functions

The types of function are:

- Named - These type of functions contains name at the time of definition. For Example:

```
1. function display()
2. {
3.   document.writeln("Named Function");
4. }
5. display();
```

- Anonymous - These type of functions doesn't contain any name. They are declared dynamically at runtime.

```
1. var display=function()
2. {
3.   document.writeln("Anonymous Function");
4. }
5. display();
```

8) Define anonymous function

It is a function that has no name. These functions are declared dynamically at runtime using the function operator instead of the function declaration. The function operator is more flexible than a function declaration. It can be easily used in the place of an expression. For example:

```
1. var display=function()
2. {
3.   alert("Anonymous Function is invoked");
4. }
5. display();
```

9) Can an anonymous function be assigned to a variable?

Yes, you can assign an anonymous function to a variable.

10) In JavaScript what is an argument object?

The variables of JavaScript represent the arguments that are passed to a function.

11) Define closure.

In JavaScript, we need closures when a variable which is defined outside the scope in reference is accessed from some inner scope.

```
1. var num = 10;
2. function sum()
3. {
4. document.writeln(num+num);
5. }
6. sum();
```

12) If we want to return the character from a specific index which method is used?

The JavaScript string charAt() method is used to find out a char value present at the specified index. The index number starts from 0 and goes to n-1, where n is the length of the string. The index value can't be a negative, greater than or equal to the length of the string. For example:

```
1. var str="Javatpoint";
2. document.writeln(str.charAt(4));
```

13) What is the difference between JavaScript and JScript?

Netscape provided the JavaScript language. Microsoft changed the name and called it JScript to avoid the trademark issue. In other words, you can say JScript is the same as JavaScript, but Microsoft provides it.

14) How to write a hello world example of JavaScript?

A simple example of JavaScript hello world is given below. You need to place it inside the body tag of HTML.

```
1. <script type="text/javascript">
2. document.write("JavaScript Hello World!");
3. </script>
```

15) What are the key differences between Java and JavaScript? / How is JavaScript different from Java?

JavaScript is a lightweight programming language (most commonly known as scripting language) developed by Netscape, Inc. It is used to make web pages interactive. It is not a part of the Java platform. Following is a list of some key differences between Java and JavaScript

A list of key differences between Java and JavaScript

Java	JavaScript
Java is a complete and strongly typed programming language used for backend coding. In Java, variables must be declared first to use in the program, and the type of a variable is checked at compile-time.	JavaScript is a weakly typed, lightweight programming language (most commonly known as scripting language) and has more relaxed s
Java is an object-oriented programming (OOPS) language or structured programming languages such as C, C++, or .Net.	JavaScript is a client-side scripting language, and it doesn't fully support the OOPS concept. It resides inside the HTML documents and is used to make web pages interactive (not achievable with simple HTML).
Java creates applications that can run in any virtual machine (JVM) or browser.	JavaScript code can run only in the browser, but it can now run on the server via Node.js.

The Java code needs to be compiled.	The JavaScript code doesn't require to be compiled.
Java Objects are class-based. You can't make any program in Java without creating a class.	JavaScript Objects are prototype-based.
Java is a Complete and Standalone language that can be used in backend coding.	JavaScript is assigned within a web page and integrates with its HTML content.
Java programs consume more memory.	JavaScript code is used in HTML web pages and requires less memory.
The file extension of the Java program is written as ".Java" and it translates source code into bytecodes which are then executed by JVM (Java Virtual Machine).	The JavaScript file extension is written as ".js" and it is interpreted but not compiled. Every browser has a JavaScript interpreter to execute the JS code.
Java supports multithreading.	JavaScript doesn't support multithreading.
Java uses a thread-based approach to concurrency.	JavaScript uses an event-based approach to concurrency.

16) How to use external JavaScript file?

I am assuming that js file name is message.js, place the following script tag inside the head tag.

1. **<script** type="text/javascript" src="message.js"**></script>**

17) Is JavaScript case sensitive language?

Yes, JavaScript is a case sensitive language. For example:

1. Var msg = "JavaScript is a case-
   sensitive language"; //Here, var should be used to declare a variable
2. function display()
3. {
4. document.writeln(msg); // It will not display the result.

5. }
6. display();

18) What is BOM?

BOM stands for *Browser Object Model*. It provides interaction with the browser. The default object of a browser is a window. So, you can call all the functions of the window by specifying the window or directly. The window object provides various properties like document, history, screen, navigator, location, innerHeight, innerWidth,

19) What is DOM? What is the use of document object?

DOM stands for *Document Object Model*. A document object represents the HTML document. It can be used to access and change the content of HTML.

20) What is the use of window object?

The window object is created automatically by the browser that represents a window of a browser. It is not an object of JavaScript. It is a browser object.

The window object is used to display the popup dialog box. Let's see with description.

Method	Description
alert()	displays the alert box containing the message with ok button.
confirm()	displays the confirm dialog box containing the message with ok and cancel button.
prompt()	displays a dialog box to get input from the user.
open()	opens the new window.
close()	closes the current window.
setTimeout()	performs the action after specified time like calling function, evaluating expressions.

21) What is the use of history object?

The history object of a browser can be used to switch to history pages such as back and forward from the current page or another page. There are three methods of history object.

1. history.back() - It loads the previous page.
2. history.forward() - It loads the next page.
3. history.go(number) - The number may be positive for forward, negative for backward. It loads the given page number.

22) How to write a comment in JavaScript?

There are two types of comments in JavaScript.

1. Single Line Comment: It is represented by // (double forward slash)
2. Multi-Line Comment: Slash represents it with asterisk symbol as /* write comment here */

23) How to create a function in JavaScript?

To create a function in JavaScript, follow the following syntax.

```
1. function function_name(){
2. //function body
3. }
```

24) What are the different data types present in JavaScript?

There are two types of data types in JavaScript:

- Primitive data types
- Non- Primitive data types

Primitive data types

The primitive data types are as follows:

String: The string data type represents a sequence of characters. It is written within quotes and can be represented using a single or a double quote.

Example:

```
1. var str1 = "Hello JavaTpoint"; //using double quotes
2. var str2 = 'Hello Javatpoint'; //using single quotes
```

Number: The number data type is used to represent numeric values and can be written with or without decimals.

Example:

```
1. var x = 5; //without decimal
2. var y = 5.0; //with decimal
```

Boolean: The Boolean data type is used to represent a Boolean value, either false or true. This data type is generally used for conditional testing.

Example:

```
1. var x = 5;
2. var y =  6;
3. var z =  5;
4. (x == y) // returns false
5. (x == z) //returns true
```

BigInt: The BigInt data type is used to store numbers beyond the Number data type limitation. This data type can store large integers and is represented by adding "n" to an integer literal.

Example:

```
1. var bigInteger =  1234567890123456789012345678901234567890;
2. // This is an example of bigInteger.
```

Undefined: The Undefined data type is used when a variable is declared but not assigned. The value of this data type is undefined, and its type is also undefined.

Example:

```
1. var x; // value of x is undefined
2. var y = undefined; // You can also set the value of a variable as undefined.
```

Null: The Null data type is used to represent a non-existent, null, or a invalid value i.e. no value at all.

Example:

```
1. var  x = null;
```

Symbol: Symbol is a new data type introduced in the ES6 version of JavaScript. It is used to store an anonymous and unique value.

Example:

```
1. var symbol1 = Symbol('symbol');
```

typeof: The typeof operator is used to determine what type of data a variable or operand contains. It can be used with or without parentheses (typeof(x) or typeof x). This is mainly used in situations when you need to process the values of different types.

Example:

```
1. typeof 10;  // Returns: "number"
2. typeof 10.0;  // Returns: "number"
3. typeof 2.5e-4;  // Returns: "number"
4. typeof Infinity;  // Returns: "number"
5. typeof NaN;  // Returns: "number". Despite being "Not-A-Number"
6. // Strings
7. typeof '';  // Returns: "string"
8. typeof 'Welcome to JavaTpoint';  // Returns: "string"
9. typeof '12';  // Returns: "string". Number within quotes is typeof string
10. // Booleans
11. typeof true;  // Returns: "boolean"
12. typeof false;  // Returns: "boolean"
13. // Undefined
14. typeof undefined;  // Returns: "undefined"
15. typeof undeclaredVariable; // Returns: "undefined"
16. // Null
17. typeof Null;  // Returns: "object"
18. // Objects
19. typeof {name: "John", age: 18};  // Returns: "object"
20. // Arrays
21. typeof [1, 2, 3];  // Returns: "object"
22. // Functions
```

```
23. typeof function(){};  // Returns: "function"
```

Non-Primitive data types

In the above examples, we can see that the primitive data types can store only a single value. To store multiple and complex values, we have to use non-primitive data types.

The non-primitive data types are as follows:

Object: The Object is a non-primitive data type. It is used to store collections of data. An object contains properties, defined as a key-value pair. A property key (name) is always a string, but the value can be any data type, such as strings, numbers, Booleans, or complex data types like arrays, functions, and other objects.

Example:

```
1. // Collection of data in key-value pairs
2. var obj1 = {
3.     x:  123,
4.     y:  "Welcome to JavaTpoint",
5.     z: function(){
6.        return this.x;
7.     }
8. }
```

Array: The Array data type is used to represent a group of similar values. Every value in an array has a numeric position, called its index, and it may contain data of any data type-numbers, strings, Booleans, functions, objects, and even other arrays. The array index starts from 0 so that the first array element is arr[0], not arr[1].

Example:

```
1. var colors = ["Red", "Yellow", "Green", "Orange"];
2. var cities = ["Noida", "Delhi", "Ghaziabad"];
3. alert(colors[2]);   // Output: Green
4. alert(cities[1]);   // Output: Delhi
```

25) What is the difference between == and ===?

The == operator checks equality only whereas === checks equality, and data type, i.e., a value must be of the same type.

26) How to write HTML code dynamically using JavaScript?

The innerHTML property is used to write the HTML code using JavaScript dynamically. Let's see a simple example:

```
1. document.getElementById('mylocation').innerHTML="<h2>This is heading us
   ing JavaScript</h2>";
```

27) How to write normal text code using JavaScript dynamically?

The innerText property is used to write the simple text using JavaScript dynamically. Let's see a simple example:

```
1. document.getElementById('mylocation').innerText="This is text using JavaScri
   pt";
```

28) How to create objects in JavaScript?

There are 3 ways to create an object in JavaScript.

1. By object literal
2. By creating an instance of Object
3. By Object Constructor

Let's see a simple code to create an object using object literal.

```
1. emp={id:102,name:"Rahul Kumar",salary:50000}
```

29) How to create an array in JavaScript?

There are 3 ways to create an array in JavaScript.

1. By array literal
2. By creating an instance of Array
3. By using an Array constructor

Let's see a simple code to create an array using object literal.

```
1. var emp=["Shyam","Vimal","Ratan"];
```

30) What does the isNaN() function?

The isNan() function returns true if the variable value is not a number. For example:

```
1. function number(num) {
```

```
    if (isNaN(num)) {
      return "Not a Number";
    }
    return "Number";
}
console.log(number('1000F'));
// expected output: "Not a Number"

console.log(number('1000'));
// expected output: "Number"
```

31) What is the output of 10+20+"30" in JavaScript?

3030 because 10+20 will be 30. If there is numeric value before and after +, it treats as binary + (arithmetic operator).

```
function display()
{
  document.writeln(10+20+"30");
}
display();
```

32) What is the output of "10"+20+30 in JavaScript?

102030 because after a string all the + will be treated as string concatenation operator (not binary +).

```
function display()
{
  document.writeln("10"+20+30);
}
display();
```

33) Difference between Client side JavaScript and Server side JavaScript?

Client-side JavaScript comprises the basic language and predefined objects which are relevant to running JavaScript in a browser. The client-side JavaScript is embedded directly by in the HTML pages. The browser interprets this script at runtime.

Server-side JavaScript also resembles client-side JavaScript. It has a relevant JavaScript which is to run in a server. The server-side JavaScript are deployed only after compilation.

34) In which location cookies are stored on the hard disk?

The storage of cookies on the hard disk depends on the OS and the browser.

The Netscape Navigator on Windows uses a cookies.txt file that contains all the cookies. The path is c:\Program Files\Netscape\Users\username\cookies.txt

The Internet Explorer stores the cookies on a file username@website.txt. The path is: c:\Windows\Cookies\username@Website.txt.

35) What's the difference between event.preventDefault() and event.stopPropagation() methods in JavaScript?

In JavaScript, the event.preventDefault() method is used to prevent the default behavior of an element.

For example: If you use it in a form element, it prevents it from submitting. If used in an anchor element, it prevents it from navigating. If used in a contextmenu, it prevents it from showing or displaying.

On the other hand, the event.stopPropagation() method is used to stop the propagation of an event or stop the event from occurring in the bubbling or capturing phase.

36) What is the real name of JavaScript?

The original name was **Mocha**, a name chosen by Marc Andreessen, founder of Netscape. In September of 1995, the name was changed to LiveScript. In December 1995, after receiving a trademark license from Sun, the name JavaScript was adopted.

37) How can you check if the event.preventDefault() method was used in an element?

When we use the event.defaultPrevent() method in the event object returns a Boolean indicating that the event.preventDefault() was called in a particular element.

38) What is the difference between undefined value and null value?

Undefined value: A value that is not defined and has no keyword is known as undefined value. For example:

```
1. int number;//Here, a number has an undefined value.
```

Null value: A value that is explicitly specified by the keyword "null" is known as a null value. For example:

1. String str=null;//Here, str has a null value.

39) How to set the cursor to wait in JavaScript?

The cursor can be set to wait in JavaScript by using the property "cursor". The following example illustrates the usage:

1. **<script>**
2. window.document.body.style.cursor = "wait";
3. **</script>**

40) What is this [[[]]]?

This is a three-dimensional array.

1. var myArray = [[[]]];

41) Are Java and JavaScript same?

No, Java and JavaScript are the two different languages. Java is a robust, secured and object-oriented programming language whereas JavaScript is a client-side scripting language with some limitations.

42) What is negative infinity?

Negative Infinity is a number in JavaScript which can be derived by dividing the negative number by zero. For example:

1. var num=-5;
2. function display()
3. {
4. document.writeln(num/0);
5. }
6. display();
7. //expected output: -Infinity

43) What is the difference between View state and Session state?

"View state" is specific to a page in a session whereas "Session state" is specific to a user or browser that can be accessed across all pages in the web application.

44) What are the pop-up boxes available in JavaScript?

- Alert Box
- Confirm Box

- Prompt Box

Example of alert() in JavaScript

```
1. <script type="text/javascript">
2. function msg(){
3.  alert("Hello Alert Box");
4. }
5. </script>
6. <input type="button" value="click" onclick="msg()"/>
```

Example of confirm() in JavaScript

```
1. <script type="text/javascript">
2. function msg(){
3. var v= confirm("Are u sure?");
4. if(v==true){
5. alert("ok");
6. }
7. else{
8. alert("cancel");
9. }
10.
11. }
12. </script>
13.
14. <input type="button" value="delete record" onclick="msg()"/>
```

Example of prompt() in JavaScript

```
1. <script type="text/javascript">
2. function msg(){
```

```
3. var v= prompt("Who are you?");
4. alert("I am "+v);
5.
6. }
7. </script>
8.
9. <input type="button" value="click" onclick="msg()"/>
```

45) How can we detect OS of the client machine using JavaScript?

The **navigator.appVersion** string can be used to detect the operating system on the client machine.

46) How to submit a form using JavaScript by clicking a link?

Let's see the JavaScript code to submit the form by clicking the link.

```
1. <form name="myform" action="index.php">
2. Search: <input type='text' name='query' />
3. <a href="javascript: submitform()">Search</a>
4. </form>
5. <script type="text/javascript">
6. function submitform()
7. {
8.   document.myform.submit();
9. }
10. </script>
```

47) Is JavaScript faster than ASP script?

Yes, because it doesn't require web server's support for execution.

48) How to change the background color of HTML document using JavaScript?

```
1. <script type="text/javascript">
2. document.body.bgColor="pink";
```

3. **</script>**

49) How to handle exceptions in JavaScript?

By the help of try/catch block, we can handle exceptions in JavaScript. JavaScript supports try, catch, finally and throw keywords for exception handling.

50) How to validate a form in JavaScript?

```
1. <script>
2. function validateform(){
3. var name=document.myform.name.value;
4. var password=document.myform.password.value;
5. 
6. if (name==null || name==""){
7.   alert("Name can't be blank");
8.   return false;
9. }else if(password.length<6){
10. alert("Password must be at least 6 characters long.");
11. return false;
12. }
13. }
14. </script>
15. <body>
16. <form name="myform" method="post" action="abc.jsp" onsubmit="return val
    idateform()" >
17. Name: <input type="text" name="name"><br/>
18. Password: <input type="password" name="password"><br/>
19. <input type="submit" value="register">
20. </form>
```

51) How to validate email in JavaScript?

```
1. <script>
2. function validateemail()
3. {
4. var x=document.myform.email.value;
5. var atposition=x.indexOf("@");
6. var dotposition=x.lastIndexOf(".");
7. if (atposition<1 || dotposition<atposition+2 || dotposition+2>=x.length){
8.   alert("Please enter a valid e-
   mail address \n atpostion:"+atposition+"\n dotposition:"+dotposition);
9.   return false;
10.  }
11. }
12. </script>
13. <body>
14. <form name="myform"  method="post" action="#" onsubmit="return validate
   email();">
15. Email: <input type="text" name="email"><br/>
16.
17. <input type="submit" value="register">
18. </form>
```

52) What is this keyword in JavaScript?

The this keyword is a reference variable that refers to the current object. For example:

```
1. var address=
2. {
3. company:"Javatpoint",
4. city:"Noida",
```

```
5. state:"UP",
6. fullAddress:function()
7. {
8. return this.company+" "+this.city+" "+this.state;
9. }
10. };
11. var fetch=address.fullAddress();
12. document.writeln(fetch);
```

53) What is the requirement of debugging in JavaScript?

JavaScript didn't show any error message in a browser. However, these mistakes can affect the output. The best practice to find out the error is to debug the code. The code can be debugged easily by using web browsers like Google Chrome, Mozilla Firebox.

To perform debugging, we can use any of the following approaches:

- Using console.log() method
- Using debugger keyword

54) What is the use of debugger keyword in JavaScript?

JavaScript debugger keyword sets the breakpoint through the code itself. The debugger stops the execution of the program at the position it is applied. Now, we can start the flow of execution manually. If an exception occurs, the execution will stop again on that particular line.. For example:

```
1. function display()
2. {
3. x = 10;
4. y = 15;
5. z = x + y;
6. debugger;
7. document.write(z);
8. document.write(a);
9. }
```

```
10. display();
```

55) What is the role of a strict mode in JavaScript?

The JavaScript strict mode is used to generates silent errors. It provides "use strict"; expression to enable the strict mode. This expression can only be placed as the first statement in a script or a function. For example:

```
1. "use strict";
2. x=10;
3. console.log(x);
```

57) What is the use of Math object in JavaScript?

The JavaScript math object provides several constants and methods to perform a mathematical operation. Unlike date object, it doesn't have constructors. For example:

```
1. function display()
2. {
3.   document.writeln(Math.random());
4. }
5. display();
```

58) What is the use of a Date object in JavaScript?

The JavaScript date object can be used to get a year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

```
1. function display()
2. {
3.   var date=new Date();
4. var day=date.getDate();
5. var month=date.getMonth()+1;
6. var year=date.getFullYear();
7. document.write("<br>Date is: "+day+"/"+month+"/"+year);
8. }
9. display();
```

59) What is the use of a Number object in JavaScript?

The JavaScript number object enables you to represent a numeric value. It may be integer or floating-point. JavaScript number object follows the IEEE standard to represent the floating-point numbers.

```
1. function display()
2. {
3. var x=102;//integer value
4. var y=102.7;//floating point value
5. var z=13e4;//exponent value, output: 130000
6. var n=new Number(16);//integer value by number object
7. document.write(x+" "+y+" "+z+" "+n);
8. }
9. display();
```

60) What is the use of a Boolean object in JavaScript?

The JavaScript Boolean is an object that represents value in two states: true or false. You can create the JavaScript Boolean object by Boolean() constructor.

```
1. function display()
2. {
3. document.writeln(10<20);//true
4. document.writeln(10<5);//false
5. }
6. display();
```

61) What is the use of a TypedArray object in JavaScript?

The JavaScript TypedArray object illustrates an array like a view of an underlying binary data buffer. There is any number of different global properties, whose values are TypedArray constructors for specific element types.

```
1. function display()
2. {
3. var arr1= [1,2,3,4,5,6,7,8,9,10];
```

```
4.      arr1.copyWithin(2) ;
5.      document.write(arr1);
6.  }
7.  display();
```

62) What is the use of a Set object in JavaScript?

The JavaScript Set object is used to store the elements with unique values. The values can be of any type i.e. whether primitive values or object references. For example:

```
1.  function display()
2.  {
3.  var set = new Set();
4.  set.add("jQuery");
5.  set.add("AngularJS");
6.  set.add("Bootstrap");
7.  for (let elements of set) {
8.   document.writeln(elements+"<br>");
9.  }
10. }
11. display();
```

63) What is the use of a WeakSet object in JavaScript?

The JavaScript WeakSet object is the type of collection that allows us to store weakly held objects. Unlike Set, the WeakSet are the collections of objects only. It doesn't contain the arbitrary values. For example:

```
1.  function display()
2.  {
3.  var ws = new WeakSet();
4.  var obj1={};
5.  var obj2={};
6.  ws.add(obj1);
```

```
7. ws.add(obj2);
8. //Let's check whether the WeakSet object contains the added object
9. document.writeln(ws.has(obj1)+"<br>");
10. document.writeln(ws.has(obj2));
11. }
12. display()
```

64) What is the use of a Map object in JavaScript?

The JavaScript Map object is used to map keys to values. It stores each element as key-value pair. It operates the elements such as search, update and delete on the basis of specified key. For example:

```
1. function display()
2. {
3. var map=new Map();
4. map.set(1,"jQuery");
5. map.set(2,"AngularJS");
6. map.set(3,"Bootstrap");
7. document.writeln(map.get(1)+"<br>");
8. document.writeln(map.get(2)+"<br>");
9. document.writeln(map.get(3));
10. }
11. display();
```

65) What is the use of a WeakMap object in JavaScript?

The JavaScript WeakMap object is a type of collection which is almost similar to Map. It stores each element as a key-value pair where keys are weakly referenced. Here, the keys are objects and the values are arbitrary values. For example:

```
1. function display()
2. {
3. var wm = new WeakMap();
```

```
4. var obj1 = {};
5. var obj2 = {};
6. var obj3= {};
7. wm.set(obj1, "jQuery");
8. wm.set(obj2, "AngularJS");
9. wm.set(obj3,"Bootstrap");
10. document.writeln(wm.has(obj2));
11. }
12. display();
```

66) What are the falsy values in JavaScript, and how can we check if a value is falsy?

Those values which become false while converting to Boolean are called falsy values.

```
1. const falsyValues = ['', 0, null, undefined, NaN, false];
```

We can check if a value is falsy by using the Boolean function or the Double NOT operator (!!).

67) What do you understand by hoisting in JavaScript?

Hoisting is the default behavior of JavaScript where all the variable and function declarations are moved on top. In simple words, we can say that Hoisting is a process in which, 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:

```
1. hoistedVariable = 12;
2. console.log(hoistedVariable); // outputs 12 even when the variable is declared a
   fter it is initialized
3. var hoistedVariable;
```

Example2:

```
1. hoistedFunction();  // Outputs " Welcome to JavaTpoint " even when the functi
   on is declared after calling
2. function hoistedFunction(){
3.   console.log(" Welcome to JavaTpoint ");
```

```
4. }
5. Example3:
6. // Hoisting in a local scope
7. function doSomething(){
8.    x = 11;
9.    console.log(x);
10.  var x;
11. }
12. doSomething(); // Outputs 11 since the local variable "x" is hoisted inside the l
    ocal scope
```

Q1. What is the difference between Java & JavaScript?

Java	JavaScript
Java is an OOP programming language.	JavaScript is an OOP scripting language.
It creates applications that run in a virtual machine or browser.	The code is run on a browser only.
Java code needs to be compiled.	JavaScript code are all in the form of text.

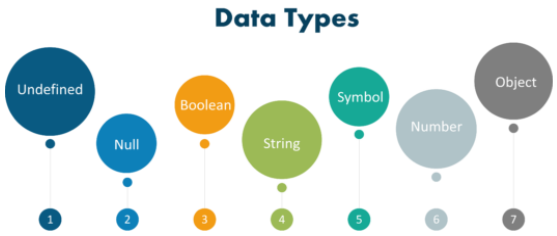
Q2. What is JavaScript?

JavaScript is a **lightweight**, **interpreted** programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

Q3. What are the data types supported by JavaScript?

The **data types** supported by JavaScript are:

- Undefined
- Null
- Boolean
- String
- Symbol
- Number
- Object



Q4. What are the features of JavaScript?

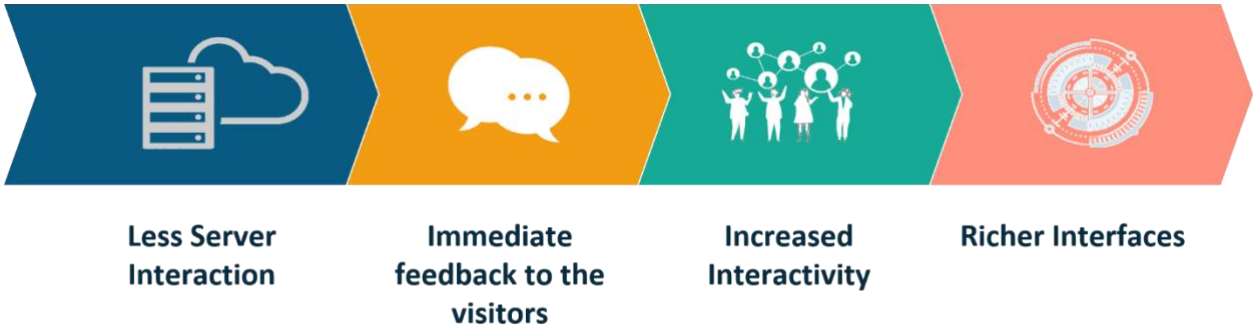
Following are the **features** of JavaScript:

- It is a **lightweight, interpreted** programming language.
- It is designed for creating **network-centric** applications.
- It is complementary to and **integrated** with Java.
- It is an **open** and **cross-platform** scripting language.

Q5. Is JavaScript a case-sensitive language?

Yes, JavaScript is a **case sensitive** language. The language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

Q6. What are the advantages of JavaScript?



Following are the **advantages** of using JavaScript −

- **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.

- **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Q7. How can you create an object in JavaScript?

JavaScript supports **Object** concept very well. You can create an object using the **object literal** as follows −

```
1var emp = {

2name: "Daniel",

3age: 23

4};
```

Q8. How can you create an Array in JavaScript?

You can define arrays using the **array literal** as follows-

```
1var x = [];

2var y = [1, 2, 3, 4, 5];
```

Q9. What is a name function in JavaScript & how to define it?

A named function declares a name as soon as it is defined. It can be defined using **function** keyword as :

```
1function named(){

2// write code here

3}
```

Q10. Can you assign an anonymous function to a variable and pass it as an argument to another function?

Yes! An anonymous function can be assigned to a variable. It can also be passed as an argument to another function.

In case you are facing any challenges with these JavaScript Interview Questions, please comment on your problems in the section below.

Q11. What is argument objects in JavaScript & how to get the type of arguments passed to a function?

JavaScript variable arguments represents the **arguments** that are passed to a function. Using **typeof** operator, we can get the type of arguments passed to a function. For example −

```
1function func(x){
2console.log(typeof x, arguments.length);
3}
4func(); //==> "undefined", 0
5func(7); //==> "number", 1
6func("1", "2", "3"); //==> "string", 3
```

Q12. What are the scopes of a variable in JavaScript?

The scope of a variable is the **region** of your program in which it is **defined**.
JavaScript variable will have only two scopes.
• **Global Variables** − A global variable has global scope which means it is visible everywhere in your JavaScript code.
• **Local Variables** − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Q13. What is the purpose of 'This' operator in JavaScript?

The JavaScript **this** keyword refers to the object it belongs to. This has different values depending on where it is used. In a method, this refers to the owner object and in a function, this refers to the global object.

Q14. What is Callback?

A **callback** is a plain JavaScript function passed to some method as an argument or option. It is a function that is to be **executed** after another function has finished executing, hence the name '**call back**'. In JavaScript, functions are objects. Because of this, functions can take functions as arguments, and can be returned by other functions.

Q15. What is Closure? Give an example.

Closures are created whenever a variable that is defined outside the **current scope** is accessed from within some inner scope. It gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created. To use a closure, simply define a function inside another function and expose it.

Q16. Name some of the built-in methods and the values returned by them.

Built-in Method	Values
CharAt()	It returns the character at the specified index.
Concat()	It joins two or more strings.
forEach()	It calls a function for each element in the array.
indexOf()	It returns the index within the calling String object of the first occurrence specified value.
length()	It returns the length of the string.
pop()	It removes the last element from an array and returns that element.
push()	It adds one or more elements to the end of an array and returns the new length of the array.
reverse()	It reverses the order of the elements of an array.

In case you are facing any challenges with these JavaScript Interview Questions, please comment on your problems in the section below.

Q17. What are the variable naming conventions in JavaScript?

The following **rules** are to be followed while **naming variables** in JavaScript:

1. You should not use any of the JavaScript **reserved keyword** as variable name. For example, break or boolean variable names are not valid.
2. JavaScript variable names should not start with a **numeral** (0-9). They must begin with a letter or the underscore character. For example, 123name is an invalid variable name but _123name or name123 is a valid one.
3. JavaScript variable names are **case sensitive**. For example, Test and test are two different variables.

Q18. How does TypeOf Operator work?

The **typeof** operator is used to get the data type of its operand. The operand can be either a **literal** or a **data structure** such as a variable, a function, or an object. It is a **unary** operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

Q19. How to create a cookie using JavaScript?

The simplest way to create a cookie is to assign a string value to the **document.cookie** object, which looks like this-

Syntax :

1document.cookie = "key1 = value1; key2 = value2; expires = date";

Q20. How to read a cookie using JavaScript?

Reading a cookie is just as simple as writing one, because the value of the document.cookie object is the cookie. So you can use this string whenever you want to access the cookie.

- The **document.cookie** string will keep a list of name = value pairs separated by semicolons, where name is the name of a cookie and value is its string value.
- You can use strings' **split()** function to break the string into key and values.

Q21. How to delete a cookie using JavaScript?

If you want to delete a cookie so that subsequent attempts to read the cookie in JavaScript return nothing, you just need to set the expiration date to a time in the past. You should define the cookie path to ensure that you delete the right cookie. Some browsers will not let you delete a cookie if you don't specify the path.

Q22. What is the difference between Attributes and Property?

Attributes- provide more details on an element like id, type, value etc.

Property- is the value assigned to the property like type="text", value='Name' etc.

Q23. List out the different ways an HTML element can be accessed in a JavaScript code.

Here are the list of ways an HTML element can be accessed in a Javascript code:
(i) **getElementById('idname'):** Gets an element by its ID name
(ii) **getElementsByClass('classname'):** Gets all the elements that have the given classname.
(iii) **getElementsByTagName('tagname'):** Gets all the elements that have the given tag name.
(iv) **querySelector():** This function takes css style selector and returns the first selected element.

Q24. In how many ways a JavaScript code can be involved in an HTML file?

There are 3 different ways in which a JavaScript code can be involved in an HTML file:

- **Inline**
- **Internal**
- **External**

An **inline** function is a JavaScript function, which is assigned to a variable created at runtime. You can differentiate between Inline Functions and Anonymous since an inline function is assigned to a variable and can be easily reused. When you need a JavaScript for a function, you can either have the script **integrated** in the page you are working on, or you can have it placed in a **separate** file that you call, when needed. This is the difference between an **internal** script and an **external** script.

Q25. What are the ways to define a variable in JavaScript?

The three possible ways of defining a variable in JavaScript are:

- **Var** – The JavaScript variables statement is used to declare a variable and, optionally, we can initialize the value of that variable. Example: var a =10; Variable declarations are processed before the execution of the code.
- **Const** – The idea of const functions is not allow them to modify the object on which they are called. When a function is declared as const, it can be called on any type of object.
- **Let** – It is a signal that the variable may be reassigned, such as a counter in a loop, or a value swap in an algorithm. It also signals that the variable will be used only in the block it's defined in.

Q26. What is a Typed language?

Typed Language is in which the values are associated with **values** and not with **variables**. It is of two types:

- **Dynamically:** in this, the variable can hold multiple types; like in JS a variable can take number, chars.
- **Statically:** in this, the variable can hold only one type, like in Java a variable declared of string can take only set of characters and nothing else.

Q27. What is the difference between Local storage & Session storage?

Local Storage – The data is not sent back to the server for every HTTP request (HTML, images, JavaScript, CSS, etc) – reducing the amount of traffic between client and server. It will stay until it is manually cleared through settings or program.

Session Storage – It is similar to local storage; the only difference is while data stored in local storage has no expiration time, data stored in session storage gets cleared when the page session ends. Session Storage will leave when the browser is closed.

In case you are facing any challenges with these JavaScript Interview Questions, please comment on your problems in the section below.

Q28. What is the difference between the operators '==' & '==='?

The main difference between "==" and "===" operator is that formerly compares variable by making **type correction** e.g. if you compare a number with a string with numeric literal, == allows that, but === doesn't allow that, because it not only checks the value but also type of two variable, if two variables are not of the same type "===" return false, while "==" return true.

Q29. What is the difference between null & undefined?

Undefined means a variable has been **declared** but has not yet been **assigned** a value. On the other hand, null is an assignment value. It can be assigned to a variable as a representation of no value. Also, undefined and null are two distinct types: undefined is a type itself (undefined) while null is an object.

Q30. What is the difference between undeclared & undefined?

Undeclared variables are those that do not **exist** in a program and are not declared. If the program tries to read the value of an undeclared variable, then a **runtime error** is encountered. Undefined variables are those that are declared in the program but have not been given any value. If the program tries to read the value of an undefined variable, an undefined value is returned.

Q31. Name some of the JavaScript Frameworks



A JavaScript framework is an application framework written in JavaScript. It differs from a JavaScript library in its control flow. There are many JavaScript Frameworks available but some of the most commonly used frameworks are:

- Angular
- React
- Vue

Q32. What is the difference between window & document in JavaScript?

Window	Document
JavaScript window is a global object which holds variables, functions, history, location.	The document also comes under the window and can be considered as the property of the window.

Q33. What is the difference between innerHTML & innerText?

innerHTML – It will process an HTML tag if found in a string

innerText – It will not process an HTML tag if found in a string

Q34. What is an event bubbling in JavaScript?

Event bubbling is a way of **event propagation** in the HTML DOM API, when an event occurs in an element inside another element, and both elements have registered a handle for that event. With bubbling, the event is first captured and handled by the **innermost** element and then propagated to outer elements. The execution starts from that event and goes to its parent element. Then the execution passes to its parent element and so on till the body element.

Q35. What is NaN in JavaScript?

NaN is a short form of **Not a Number.** Since NaN always compares unequal to any number, including NaN, it is usually used to indicate an error condition for a function that should return a valid number. When a string or something else is being **converted** into a **number** and that cannot be done, then we get to see NaN.

In case you are facing any challenges with these JavaScript Interview Questions, please comment on your problems in the section below.

Q36. How do JavaScript primitive/object types passed in functions?

One of the differences between the two is that Primitive Data Types are passed By Value and Objects are passed By Reference.

- **By Value** means creating a COPY of the original. Picture it like twins: they are born exactly the same, but the first twin doesn't lose a leg when the second twin loses his in the war.
- **By Reference** means creating an ALIAS to the original. When your Mom calls you "Pumpkin Pie" although your name is Margaret, this doesn't suddenly give birth to a clone of yourself: you are still one, but you can be called by these two very different names.

Q37. How can you convert the string of any base to integer in JavaScript?

The **parseInt()** function is used to convert numbers between different bases. It takes the string to be converted as its first parameter, and the second parameter is the base of the given string.

For example-

```
1parseInt("4F", 16)
```

Q38. What would be the result of 2+5+"3″?

Since 2 and 5 are integers, they will be added numerically. And since 3 is a string, its concatenation will be done. So the result would be 73. The " " makes all the difference here and represents 3 as a string and not a number.

Q39. What are Exports & Imports?

Imports and exports help us to write modular JavaScript code. Using Imports and exports we can split our code into multiple files. For example-

```
1  //------ lib.js ------</span>
2  export const sqrt = Math.sqrt;</span>
3  export function square(x) {</span>
4  return x * x;</span>
5  }
```

```
6  export function diag(x, y) {
7  return sqrt(square(x) + square(y));
8  }
9
10//------ main.js ------</span>
11 { square, diag } from 'lib';
12console.log(square(5)); // 25
13console.log(diag(4, 3)); // 5
```

Now with this, we have reached the final section of JS Interview Questions.

Advanced Level JavaScript Interview Questions and Answers for Experienced Professionals

Q40. What is the 'Strict' mode in JavaScript and how can it be enabled?

Strict mode is a way to introduce better error-checking into your code.

- When you use strict mode, you cannot use implicitly declared variables, or assign a value to a read-only property, or add a property to an object that is not extensible.
- You can enable strict mode by adding "use strict" at the beginning of a file, a program, or a function.

Q41. What is a prompt box in JavaScript?

A prompt box is a box which allows the user to enter input by providing a **text box**. The prompt() method displays a dialog box that prompts the visitor for input. A prompt box is often used if you want the user to input a value before entering a page. When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

Q42. What will be the output of the code below?

```
1var Y = 1;
2if (function F(){})
3{
4y += Typeof F;</span>
5}
6console.log(y);
```

The output would be 1undefined. The if condition statement evaluates using eval, so eval(function f(){}) returns function f(){} (which is true). Therefore, inside the if

statement, executing typeof f returns undefined because the if statement code executes at run time, and the statement inside the if condition is evaluated during run time.

Q43. What is the difference between Call & Apply?

The **call()** method calls a function with a given this value and arguments provided individually.

Syntax-

```
1fun.call(thisArg[, arg1[, arg2[, ...]]])
```

The **apply()** method calls a function with a given this value, and arguments provided as an array.

Syntax-

```
1fun.apply(thisArg, [argsArray])
```

Q44. How to empty an Array in JavaScript?

There are a number of methods you can use to **empty** an **array**:

Method 1 –

```
1arrayList = []
```

Above code will set the variable arrayList to a new empty array. This is recommended if you don't have references to the original array arrayList anywhere else, because it will actually create a new, empty array. You should be careful with this method of emptying the array, because if you have referenced this array from another variable, then the original reference array will remain unchanged.

Method 2 –

```
1arrayList.length = 0;
```

The code above will clear the existing array by setting its length to 0. This way of emptying the array also updates all the reference variables that point to the original array. Therefore, this method is useful when you want to update all reference variables pointing to arrayList.

Method 3 –

```
1arrayList.splice(0, arrayList.length);
```

The implementation above will also work perfectly. This way of emptying the array will also update all the references to the original array.

Method 4 –

```
1while(arrayList.length)
2{
3arrayList.pop();
```

```
4}
```

The implementation above can also empty arrays, but it is usually not recommended to use this method often.

Q45. What will be the output of the following code?

```
1var Output = (function(x)

2{

3Delete X;

4return X;

5}

6)(0);

7console.log(output);
```

The output would be 0. The delete operator is used to delete properties from an object. Here x is not an object but a local variable. delete operators don't affect local variables.

In case you are facing any challenges with these JavaScript Interview Questions, please comment on your problems in the section below.

Q46. What will be the output of the following code?

```
1var X = { Foo : 1};

2var Output = (function()

3{

4delete X.foo;

5return X.foo;

6}

7)();

8console.log(output);
```

The output would be undefined. The delete operator is used to delete the property of an object. Here, x is an object which has the property foo, and as it is a self-invoking function, we will delete the foo property from object x. After doing so, when we try to reference a deleted property foo, the result is undefined.

Q47. What will be the output of the following code?

```
1var Employee =

2{

3company: 'xyz'
```

4}

5var Emp1 = Object.create(employee);

6delete Emp1.company Console.log(emp1.company);

The output would be xyz. Here, emp1 object has company as its prototype property. The delete operator doesn't delete prototype property. emp1 object doesn't have company as its own property. However, we can delete the company property directly from the Employee object using delete Employee.company.

Q48. What will be the output of the code below?

1//nfe (named function expression)

2var Foo = Function Bar()

3{

4return 7;

5};

6typeof Bar();

The output would be Reference Error. A function definition can have only one reference variable as its function name.

Q49. What is the reason for wrapping the entire content of a JavaScript source file in a function book?

This is an increasingly common practice, employed by many popular JavaScript libraries. This technique creates a closure around the entire contents of the file which, perhaps most importantly, creates a private namespace and thereby helps avoid potential name clashes between different JavaScript modules and libraries. Another feature of this technique is to allow for an easy alias for a global variable. This is often used in jQuery plugins.

Q50. What are escape characters in JavaScript?

JavaScript escape characters enable you to write special characters without breaking your application. Escape characters (Backslash) is used when working with special characters like single quotes, double quotes, apostrophes and ampersands. Place backslash before the characters to make it display.

Example:

1document.write "I am a "good" boy"

2document.write "I am a "good" boy"

1. What is JavaScript?

JavaScript is a client-side and server-side scripting language inserted into HTML pages and is understood by web browsers. JavaScript is also an Object-based Programming language

2. Enumerate the differences between Java and JavaScript?

Java is a complete programming language. In contrast, JavaScript is a coded program that can be introduced to HTML pages. These two languages are not at all inter-dependent and are designed for different intent. Java is an object-oriented programming (OOPS) or structured programming languages like C++ or C, whereas JavaScript is a client-side scripting language.

3. What are JavaScript Data Types?

Following are the JavaScript Data types:

- Number
- String
- Boolean
- Object
- Undefined

4. What is the use of isNaN function?

isNan function returns true if the argument is not a number; otherwise, it is false.

5. Which is faster between JavaScript and an ASP script?

JavaScript is faster. JavaScript is a client-side language,, and thus it does not need the assistance of the webserver to execute. On the other hand, ASP is a server-side language and hence is always slower than JavaScript. Javascript now is also a server-side language (nodejs).

6. What is negative Infinity?

Negative Infinity is a number in JavaScript which can be derived by dividing negative number by zero.

7. Is it possible to break JavaScript Code into several lines?

Breaking within a string statement can be done by using a backslash, '\,' at the end of the first line.

Example:

document. Write ("This is \a program,");

And if you change to a new line when not within a string statement, then javaScript ignores the break in the line.

Example:

var x=1, y=2,

z=

x+y;

The above code is perfectly fine, though not advisable as it hampers debugging.

8. Which company developed JavaScript?

Netscape is the software company that developed JavaScript.

9. What are undeclared and undefined variables?

Undeclared variables are those that do not exist in a program and are not declared. If the program tries to read the value of an undeclared variable, then a runtime error is encountered.

Undefined variables are those that are declared in the program but have not been given any value. If the program tries to read the value of an undefined variable, an undefined value is returned.

10. Write the code for adding new elements dynamically?

```
<html>

<head>

<title>t1</title>

<script type="text/javascript">

   function addNode () { var newP = document. createElement("p");

   var textNode = document.createTextNode(" This is a new text node");

   newP.appendChild(textNode);
document.getElementById("firstP").appendChild(newP); }

</script> </head>

<body> <p id="firstP">firstP<p> </body>

</html>
```

11. What are global variables? How are these variable declared?

Global variables are available throughout the length of the code so that it has no scope. The var keyword is used to declare a local variable or object. If the var keyword is omitted, a global variable is declared.

Example:

// Declare a global: globalVariable = “Test”;

The problems faced by using global variables are the clash of variable names of local and global scope. Also, it is difficult to debug and test the code that relies on global variables.

12. What is a prompt box?

A prompt box is a box that allows the user to enter input by providing a text box. A label and box will be provided to enter the text or number.

13. What is 'this' keyword in JavaScript?

'This' keyword refers to the object from where it was called.

14. What is the working of timers in JavaScript?

Timers are used to execute a piece of code at a set time or repeat the code in a given interval. This is done by using the functions **setTimeout, setInterval,** and **clearInterval**.

The **setTimeout(function, delay)** function is used to start a timer that calls a particular function after the mentioned delay. The **setInterval(function, delay)** function repeatedly executes the given function in the mentioned delay and only halts when canceled. The **clearInterval(id)** function instructs the timer to stop.

Timers are operated within a single thread, and thus events might queue up, waiting to be executed.

15. Which symbol is used for comments in Javascript?

// for Single line comments and

/* Multi

Line

Comment

*/

16. What is the difference between ViewState and SessionState?

- 'ViewState' is specific to a page in a session.
- 'SessionState' is specific to user-specific data that can be accessed across all web application pages.

17. What is === operator?

=== is called a strict equality operator, which returns true when the two operands have the same value without conversion.

18. How you can submit a form using JavaScript?

To submit a form using JavaScript use

document.form[0].submit();

document.form[0].submit();

19. Does JavaScript support automatic type conversion?

Yes, JavaScript does support automatic type conversion. It is the common way of type conversion used by JavaScript developers

20. How can the style/class of an element be changed?

It can be done in the following way:

document.getElementById("myText"). style. fontSize = "20";

or

document. getElementById ("myText"). className = "anyclass";

21. How to read and write a file using JavaScript?

There are two ways to read and write a file using JavaScript

- Using JavaScript extensions
- Using a web page and Active X objects

22. What are all the looping structures in JavaScript?

Following are looping structures in Javascript:

- For
- While
- Do-while loops

23. What is called Variable typing in Javascript?

Variable typing is used to assign a number to a variable. The same variable can be assigned to a string.

Example:

i = 10;

i = "string;"

This is called variable typing.

24. How can you convert the string of any base to an integer in JavaScript?

The parseInt() function is used to convert numbers between different bases. parseInt() takes the string to be converted as its first parameter. The second parameter is the base of the given string.

To convert 4F (or base 16) to integer, the code used will be –

parseInt ("4F", 16);

25. Difference between "==" and "==="?

"==" checks only for equality in value, whereas "===" is a stricter equality test and returns false if either the value or the type of the two variables are different.

26. What would be the result of 3+2+"7"?

Since 3 and 2 are integers, they will be added numerically. And since 7 is a string, its concatenation will be done. So the result would be 57.

27. How to detect the operating system on the client machine?

In order to detect the operating system on the client machine, the navigator. Platform string (property) should be used.

28. What do you mean by NULL in Javascript?

The NULL value is used to represent no value or no object. It implies no object or null string, no valid boolean value, no number, and no array object.

29. What is the function of the delete operator?

The delete keyword is used to delete the property as well as its value.

Example

var student= {age:20, batch:"ABC"};

Delete student. age;

30. What is an undefined value in JavaScript?

Undefined value means the

- Variable used in the code doesn't exist
- Variable is not assigned to any value
- Property does not exist.

31. What are all the types of Pop up boxes available in JavaScript?

- Alert
- Confirm and
- Prompt

32. What is the use of Void (0)?

Void(0) is used to prevent the page from refreshing, and parameter "zero" is passed while calling.

Void(0) is used to call another method without refreshing the page.

33. How can a page be forced to load another page in JavaScript?

The following code has to be inserted to achieve the desired effect:

<script language="JavaScript" type="text/javascript" >

<!-- location. href="https://www.guru99.com/javascript-interview-questions-answers.html"; //--></script>

34. What is the data type of variables in JavaScript?

All variables in JavaScript are object data types.

35. What is the difference between an alert box and a confirmation box?

An alert box displays only one button, which is the OK button.

But a Confirmation box displays two buttons, namely OK and cancel.

36. What are escape characters?

Escape characters (Backslash) is used when working with special characters like single quotes, double quotes, apostrophes, and ampersands. Place backslash before the characters to make it display.

Example:

document. write "I m a "good" boy."

document. write "I m a \"good\" boy."

37. What are JavaScript Cookies?

Cookies are the small test files stored in a computer, and they get created when the user visits the websites to store information that they need. Examples could be User Name details and shopping cart information from previous visits.

38. What a pop()method in JavaScript is?

The pop() method is similar to the shift() method, but the difference is that the Shift method works at the array's start. The pop() method takes the last element off of the given array and returns it. The array on which it is called is then altered.

Example:

var cloths = ["Shirt", "Pant", "TShirt"];

cloths.pop();

//Now cloth becomes Shirt,Pant

39. Does JavaScript has concept level scope?

No. JavaScript does not have concept-level scope. The variable declared inside the function has scope inside the function.

40. What are the disadvantages of using innerHTML in JavaScript?

If you use innerHTML in JavaScript, the disadvantage is

- Content is replaced everywhere

- We cannot use it like “appending to innerHTML
- Even if you use +=like “innerHTML = innerHTML + ‘html’” still the old content is replaced by html
- The entire innerHTML content is re-parsed and builds into elements. Therefore, it’s much slower
- The innerHTML does not provide validation, and therefore we can potentially insert valid and broken HTML in the document and break it

41. What is break and continue statements?

Break statement exits from the current loop.

Continue statement continues with next statement of the loop.

42. What are the two basic groups of data types in JavaScript?

- They are as—Primitive
- Reference types

Primitive types are number and Boolean data types. Reference types are more complex types like strings and dates.

43. How can generic objects be created?

Generic objects can be created as:

```
var I = new object();
```

44. What is the use of a type of operator?

‘Typeof’ is an operator used to return a string description of the type of a variable.

45. Which keywords are used to handle exceptions?

Try… Catch—finally is used to handle exceptions in the JavaScript

```
Try{
   Code
}
Catch(exp){
   Code to throw an exception.
}
Finally{
   Code runs either it finishes successfully or after catch
}
```

46. Which keyword is used to print the text on the screen?

Document. Write ("Welcome") is used to print the text–Welcome on the screen.

47. What is the use of the blur function?

Blur function is used to remove the focus from the specified object.

48. What is variable typing?

Variable typing assigns a number to a variable and then assigns a string to the same variable. An example is as follows:

i= 8;

i="john";

49. How to find an operating system in the client machine using JavaScript?

The 'Navigator. the app version is used to find the operating system's name in the client machine.

50. What are the different types of errors in JavaScript?

There are three types of errors:

- **Load time errors**: Errors that come up when loading a web page, like improper syntax errors, are known as Load time errors and generate the errors dynamically.
- **Runtime errors**: Errors that come due to misuse of the command inside the HTML language.
- **Logical Errors**: These are the errors that occur due to the bad logic performed on a function with a different operation.

51. What is the use of the Push method in JavaScript?

The push method is used to add or append one or more elements to an Array end. Using this method, we can append multiple elements by passing multiple arguments.

52. What is the unshift method in JavaScript?

Unshift method is like the push method, which works at the beginning of the array. This method is used to prepend one or more elements to the beginning of the array.

53. What is the difference between JavaScript and Jscript?

Both are almost similar. Netscape and Jscript develop JavaScript was developed by Microsoft.

54. How are object properties assigned?

Properties are assigned to objects in the following way –

obj ["class"] = 12;

or

obj.class = 12;

55. What is the 'Strict Mode in JavaScript, and how can it be enabled?

Strict Mode adds certain compulsions to JavaScript. Under the strict Mode, JavaScript shows errors for a piece of code, which did not show an error before, but might be problematic and potentially unsafe. Strict Mode also solves some mistakes that hamper the JavaScript engines from working efficiently.

Strict mode can be enabled by adding the string literal "use strict" above the file. This can be illustrated by the given example:

```
function myfunction() {
   "use strict;"
   var v = "This is a strict mode function";
}
```

56. What is the way to get the status of a CheckBox?

The status can be acquired as follows –

```
alert(document.getElementById('checkbox1').checked);
```

If the CheckBox is checked, this alert will return TRUE.

57. How can the OS of the client machine be detected?

The navigator. appVersion string can be used to detect the operating system on the client machine.

58. What is a window.onload and onDocumentReady?

The onload function is not run until all the information on the page is loaded. This leads to a substantial delay before any code is executed.

onDocumentReady loads the code just after the DOM is loaded. This allows early manipulation of the code.

59. How closures work in JavaScript?

The closure is a locally declared variable related to a function that stays in memory when it has returned.

For example:

```
function greet(message) {
   console.log(message);
}
function greeter(name, age) {
```

```
    return name + " says howdy!! He is " + age + " years old";
}
// Generate the message
var message = greeter("James", 23);
// Pass it explicitly to greet
greet(message);
```

This function can be better represented by using closures

```
function greeter(name, age) {
    var message = name + " says howdy!! He is " + age + " years old";
    return function greet() {
        console.log(message);
    };
}
// Generate the closure
var JamesGreeter = greeter("James", 23);
// Use the closure
JamesGreeter();
```

60. How can a value be appended to an array?

A value can be appended to an array in the given manner –

```
arr[arr.length] = value;
```

61. What is for-in loop in Javascript?

The for-in loop is used to loop through the properties of an object.

The syntax for the for-in loop is –

```
for (variable name in object){
    statement or block to execute
}
```

In each repetition, one property from the object is associated with the variable name. The loop is continued till all the properties of the object are depleted.

62. What are the important properties of an anonymous function in JavaScript?

A function that is declared without any named identifier is known as an anonymous function. In general, an anonymous function is inaccessible after its declaration.

Anonymous function declaration –

```
var anon = function() {
   alert('I am anonymous');
};
anon();
```

63. What is the difference between .call() and .apply()?

The function .call() and .apply() are very similar in their usage except a little difference. .call() is used when the number of the function's arguments are known to the programmer, as they have to be mentioned as arguments in the call statement. On the other hand, .apply() is used when the number is not known. The function .apply() expects the argument to be an array.

The basic difference between .call() and .apply() is in the way arguments are passed to the function. Their usage can be illustrated by the given example.

```
var someObject = {
myProperty : 'Foo',

myMethod : function(prefix, postfix) {

   alert(prefix + this.myProperty + postfix);
}
};
someObject.myMethod('<', '>'); // alerts '<Foo>'
var someOtherObject  = {

   myProperty : 'Bar.'

};
someObject.myMethod.call(someOtherObject, '<', '>'); // alerts '<Bar>'
```

someObject.myMethod.apply(someOtherObject, ['<', '>']); // alerts '<Bar>'

64. What is event bubbling?

JavaScript allows DOM elements to be nested inside each other. In such a case, if the handler of the child is clicked, the handler of the parent will also work as if it were clicked too.

65. Is JavaScript case sensitive? Give its example.

Yes, JavaScript is case-sensitive. For example, a function parseInt is not the same as the function Parseint.

66. What boolean operators can be used in JavaScript?

The 'And' Operator (&&), 'Or' Operator (||), and the 'Not' Operator (!) can be used in JavaScript.

*Operators are without the parenthesis.

67. How can a particular frame be targeted, from a hyperlink, in JavaScript?

This can be done by including the name of the required frame in the hyperlink using the 'target' attribute.

>New Page

68. What is the role of break and continue statements?

The break statement is used to come out of the current loop. In contrast, the continue statement continues the current loop with a new recurrence.

69. Write the point of difference between a web garden and a web farm?

Both web-garden and web-farm are web hosting systems. The only difference is that web-garden is a setup that includes many processors in a single server. At the same time,web-farm is a larger setup that uses more than one server.

70. How are object properties assigned?

Assigning properties to objects is done in the same way as a value is assigned to a variable. For example, a form object's action value is assigned as 'submit' in the following manner – Document. form.action="submit"

71. What is the method for reading and writing a file in JavaScript?

This can be done by Using JavaScript extensions (runs from JavaScript Editor), for example, for the opening of a file –

fh = fopen(getScriptPath(), 0);

72. How are DOM utilized in JavaScript?

DOM stands for Document Object Model and is responsible for how various objects in a document interact with each other. DOM is required for developing web pages, which includes objects like paragraphs, links, etc. These objects can be operated to include

actions like add or delete. DOM is also required to add extra capabilities to a web page. On top of that, the use of API gives an advantage over other existing models.

73. How are event handlers utilized in JavaScript?

Events are the actions that result from activities, such as clicking a link or filling a form by the user. An event handler is required to manage the proper execution of all these events. Event handlers are an extra attribute of the object. This attribute includes the event's name and the action taken if the event takes place.

74. What is the role of deferred scripts in JavaScript?

The HTML code's parsing during page loading is paused by default until the script has not stopped executing. If the server is slow or the script is particularly heavy, then the web page is delayed.

While using Deferred, scripts delays execution of the script till the time the HTML parser is running. This reduces the loading time of web pages, and they get displayed faster.

75. What are the various functional components in JavaScript?

The different functional components in JavaScript are-

- **First-class functions:** Functions in JavaScript are utilized as first-class objects. This usually means that these functions can be passed as arguments to other functions, returned as values from other functions, assigned to variables, or can also be stored in data structures.
- **Nested functions:** The functions, which are defined inside other functions, are called Nested functions. They are called 'every time the main function is invoked.

76. Write about the errors shown in JavaScript?

JavaScript gives a message as if it encounters an error. The recognized errors are –

- **Load-time errors:** The errors shown at the time of the page loading are counted under Load-time errors. The use of improper syntax encounters these errors and is thus detected while the page is getting loaded.
- **Runtime errors:** This is the error that comes up while the program is running. For example, illegal operations cause the division of a number by zero or access a non-existent area of the memory.
- **Logic errors:** It is caused by syntactically correct code, which does not fulfill the required task—for example, an infinite loop.

77. What are Screen objects?

Screen objects are used to read the information from the client's screen. The properties of screen objects are –

- AvailHeight: Gives the height of the client's screen

- AvailWidth: Gives the width of the client's screen
- ColorDepth: Gives the bit depth of images on the client's screen
- Height: Gives the total height of the client's screen, including the taskbar
- Width: Gives the total width of the client's screen, including the taskbar

78. What is the unshift() method?

This method is functional at the starting of the array, unlike the push(). It adds the desired number of elements to the top of an array. For example –

```
var name = [ "john" ];

name.unshift( "charlie" );

name.unshift( "joseph", "Jane" );

console.log(name);
```

The output is shown below:

[" joseph ","," Jane ,", " charlie ", " john "]

79. What is unescape() and escape() functions?

The escape () function is responsible for coding a string to transfer the information from one computer to the other across a network.

For Example:

```
<script>

document.write(escape("Hello? How are you!"));

</script>
```

Output: Hello%3F%20How%20are%20you%21

The unescape() function is very important as it decodes the coded string.

It works in the following way. For example:

```
<script>

   document.write(unescape("Hello%3F%20How%20are%20you%21"));

</script>
```

Output: Hello? How are you!

80. What are the decodeURI() and encodeURI()?

EncodeURl() is used to convert URL into their hex coding. And DecodeURI() is used to convert the encoded URL back to normal.

```
<script>
```

```
var uri="my test.asp?name=ståle&car=saab";

document.write(encodeURI(uri)+ "<br>");

document.write(decodeURI(uri));
</script>
```

Output –

my%20test.asp?name=st%C3%A5le&car=saab

my test.asp?name=ståle&car=saab

81. Why you should not use innerHTML in JavaScript?

innerHTML content is refreshed every time and thus is slower. There is no scope for validation in innerHTML. Therefore, it is easier to insert rogue code in the document and make the web page unstable.

82. What does the following statement declare?

var myArray = [[[]]];

It declares a three-dimensional array.

83. How are JavaScript and ECMA Script related?

ECMA Script is like rules and guidelines, while Javascript is a scripting language used for web development.

84. What is namespacing in JavaScript, and how is it used?

Namespacing is used for grouping the desired functions, variables, etc., under a unique name. It is a name that has been attached to the desired functions, objects, and properties. This improves modularity in the coding and enables code reuse.

85. How can JavaScript codes be hidden from old browsers that do not support JavaScript?

For hiding JavaScript codes from old browsers:

Add "<!–" without the quotes in the code just after the <script> tag.

Add "//–>" without the quotes in the code just before the <script> tag.

Old browsers will now treat this JavaScript code as a long HTML comment. While a browser that supports JavaScript will take the "<!–" and "//–>" as one-line comments.

86. How to use Loop in JavaScript?

Loops are useful when you repeatedly execute the same lines of code a specific number of times or as long as a specific condition is true. Suppose you want to type a 'Hello'

message 100 times on your webpage. Of course, you will have to copy and paste the same line 100 times. Instead, if you use loops, you can complete this task in just 3 or 4 lines.

88. What are the important JavaScript Array Method explain with example?

JavaScript Array Methods

The Array object has many properties and methods which help developers to handle arrays easily and efficiently. You can get the value of a property by specifying arrayname.property and the output of a method by specifying arrayname.method().

- **length property** –> If you want to know the number of elements in an array, you can use the length property.
- **prototype property** –> If you want to add new properties and methods, you can use the prototype property.
- **reverse method** –> You can reverse the order of items in an array using a reverse method.
- **sort method –>** You can sort the items in an array using sort method.
- **pop method** –> You can remove the last item of an array using a pop method.
- **shift method** –> You can remove the first item of an array using shift method.
- **push method** –> You can add a value as the last item of the array.

90. What is Loop Though the Properties of an Object?

The for/in a loop is usually used to loop through the properties of an object. You can give any name for the variable, but the object's name should be the same as an already existing object you need to loop through.

Syntax:

for (variablename in objectname)

{

lines of code to be executed

}

Example:

<html>

<head>

```
<script type="text/javascript">
    var employee={first:"John", last:"Doe", department:"Accounts"};
                var details = "";
                document.write("<b>Using for/in loops </b><br />");
    for (var x in employee)
    {
                details = x + ": " + employee[x];
                document.write(details + "<br />");
    }
</script>
</head>
<body>
</body>
</html>
```

91. What is JavaScript Unit Testing, and what are the challenges in JavaScript Unit Testing?

JavaScript Unit Testing is a testing method in which JavaScript tests code written for a web page or web application module. It is combined with HTML as an inline event handler and executed in the browser to test if all functionalities work fine. These unit tests are then organized in the test suite.

Every suite contains several tests designed to be executed for a separate module. Most importantly, they don't conflict with any other module and run with fewer dependencies on each other (some critical situations may cause dependencies).

Challenges of JavaScript Unit Testing:

Here are important challenges of JavaScript Unit Testing:

- Many other languages support unit testing in browsers, in the stable as well as in runtime environment, but JavaScript can not
- You can understand some system actions with other languages, but this is not the case with JavaScript
- Some JavaScript are written for a web application that may have multiple dependencies.
- JavaScript is good to use in combination with HTML and CSS rather than on the web
- Difficulties with page rendering and DOM manipulation

- Sometimes you find an error message on your screen regarding ‘Unable to load example.js’ or any other JavaScript error regarding version control. These vulnerabilities come under Unit Testing JavaScript

Solutions of JavaScript Unit Testing:

To avoid such issues, what you can do is;

- Do not use global variables.
- Do not manipulate predefined objects.
- Design core functionalities based on the library.
- Try to create small pieces of functionalities with lesser dependencies.

92. What are some important JavaScript Unit Testing Frameworks?

Following is a curated list of popular JavaScript Unit Testing Frameworks and Tools that are widely used :

Unit.js: It is known as an open-source assertion library running on browser and Node.js. It is extremely compatible with other JavaScript Unit Testing frameworks like Mocha, Karma, Jasmine, QUnit, Protractor, etc. Provides the full documented API of assertion list.

QUnit: It is used for both client-side and server-side JavaScript Unit Testing. This Free JavaScript testing framework is used for jQuery projects. It follows Common JS unit testing Specification for unit testing in JavaScript. It supports the Node Long-term Support Schedule.

Jasmine: Jasmine is the behavior-driven development framework to unit test JavaScript. It is used for testing both synchronous and asynchronous JavaScript codes. It does not require DOM and comes with an easy syntax that can be written for any test.

Karma: Karma is an open-source productive testing environment. Easy workflow control running on the command line. Offers the freedom to write the tests with Jasmine, Mocha, and QUnit. You can run the test on real devices with easy debugging.

Mocha: Mocha runs on Node.js and in the browser. Mocha performs asynchronous testing more simply. Provides accuracy and flexibility in reporting. Provides tremendous support of rich features such as test-specific timeouts, JavaScript APIs.

Jest: Facebook uses jest so far to test all the JavaScript code. It provides the ‘zero-configuration testing experience. Supports independent and non-interrupting running tests without any conflict. Do not require any other setup configuration and libraries.

AVA: AVA is a simple JavaScript Unit Testing Framework. Tests are being run in parallel and serially. Parallel tests run without interrupting each other. This testing framework supports asynchronous testing as well. AVA uses subprocesses to run the unit test JavaScript.

94. What is DOM in JavaScript?

JavaScript can access all the elements in a web page using the Document Object Model (DOM). The web browser creates a DOM of the webpage when the page is loaded.

95. How to use DOM and Events?

Using DOM, JavaScript can perform multiple tasks. It can create new elements and attributes, change the existing elements and attributes and even remove existing elements and attributes. JavaScript can also react to existing events and create new events in the page.

97. When to Use Internal and External JavaScript Code?

Suppose you have only a few lines of code that is specific to a particular webpage. In that case, it is better to keep your JavaScript code internal within your HTML document.

On the other hand, if your JavaScript code is used in many web pages, you should consider keeping your code in a separate file. If you wish to make some changes to your code, you have to change only one file, making code maintenance easy. If your code is too long, it is better to keep it in a separate file. This helps in easy debugging.

98. What are Cookies in JavaScript?

A cookie is a piece of data stored on your computer to be accessed by your browser. You also might have enjoyed the benefits of cookies knowingly or unknowingly. Have you ever saved your Facebook password so that you do not have to type it every time you try to login? If yes, then you are using cookies. Cookies are saved as key/value pairs.

1. What do you understand about JavaScript?

JavaScript is a popular web scripting language and is used for client-side and server-side development. The JavaScript code can be inserted into HTML pages that can be understood and executed by web browsers while also supporting object-oriented programming abilities.

2. What's the difference between JavaScript and Java?

JavaScript	Java
JavaScript is an object-oriented scripting language.	Java is an object-oriented programming language.

JavaScript applications are meant to run inside a web browser.	Java applications are generally made for use in operating systems and virtual machines.
JavaScript does not need compilation before running the application code.	Java source code needs a compiler before it can be ready to run in realtime.

3. 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

4. What are the features of JavaScript?

These are the features of JavaScript:

- Lightweight, interpreted programming language
- Cross-platform compatible
- Open-source
- Object-oriented
- Integration with other backend and frontend technologies
- Used especially for the development of network-based applications

5. What are the advantages of JavaScript over other web technologies?

These are the advantages of JavaScript:

Enhanced Interaction

JavaScript adds interaction to otherwise static web pages and makes them react to users' inputs.

Quick Feedback

There is no need for a web page to reload when running JavaScript. For example, form input validation.

Rich User Interface

JavaScript helps in making the UI of web applications look and feel much better.

Frameworks

JavaScript has countless frameworks and libraries that are extensively used for developing web applications and games of all kinds.

6. How do you create an object in JavaScript?

Since JavaScript is essentially an object-oriented scripting language, it supports and encourages the usage of objects while developing web applications.

```
const student = {
    name: 'John',
    age: 17
}
```

7. How do you create an array in JavaScript?

Here is a very simple way of creating arrays in JavaScript using the array literal:

```
var a = [];
var b = ['a', 'b', 'c', 'd', 'e'];
```

8. What are some of the built-in methods in JavaScript?

Built-in Method	Values
Date()	Returns the present date and time

concat()	Joins two strings and returns the new string
push()	Adds an item to an array
pop()	Removes and also returns the last element of an array
round()	Rounds of the value to the nearest integer and then returns it
length()	Returns the length of a string

9. What are the scopes of a variable in JavaScript?

The scope of a variable implies where the variable has been declared or defined in a JavaScript program. There are two scopes of a variable:

Global Scope

Global variables, having global scope are available everywhere in a JavaScript code.

Local Scope

Local variables are accessible only within a function in which they are defined.

10. What is the 'this' keyword in JavaScript?

The 'this' keyword in JavaScript refers to the currently calling object. It is commonly used in constructors to assign values to object properties.

11. What are the conventions of naming a variable in JavaScript?

Following are the naming conventions for a variable in JavaScript:

- Variable names cannot be similar to that of reserved keywords. For example, var, let, const, etc.
- Variable names cannot begin with a numeric value. They must only begin with a letter or an underscore character.
- Variable names are case-sensitive.

12. What is Callback in JavaScript?

In JavaScript, functions are objects and therefore, functions can take other functions as arguments and can also be returned by other functions.

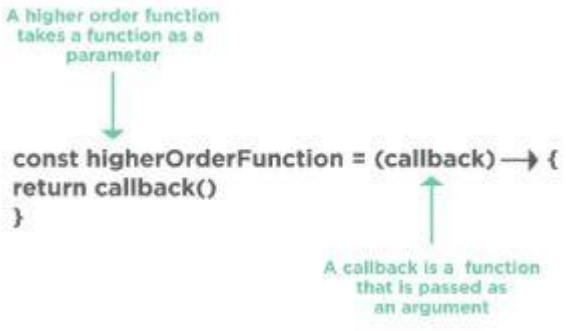


Fig: Callback function

A callback is a JavaScript function that is passed to another function as an argument or a parameter. This function is to be executed whenever the function that it is passed to gets executed.

13. How do you debug a JavaScript code?

All modern web browsers like Chrome, Firefox, etc. have an inbuilt debugger that can be accessed anytime by pressing the relevant key, usually the F12 key. There are several features available to users in the debugging tools.

We can also debug a JavaScript code inside a code editor that we use to develop a JavaScript application—for example, Visual Studio Code, Atom, Sublime Text, etc.

14. What is the difference between Function declaration and Function expression?

Function declaration	Function expression
Declared as a separate statement within the main JavaScript code	Created inside an expression or some other construct
Can be called before the function is defined	Created when the execution point reaches it; can be used only after that

Offers better code readability and better code organization	Used when there is a need for a conditional declaration of a function

Example:
function abc() {
return 5;
}
	Example:
	var a = function abc() {
	return 5;
	}

15. What are the ways of adding JavaScript code in an HTML file?

There are primarily two ways of embedding JavaScript code:

- We can write JavaScript code within the script tag in the same HTML file; this is suitable when we need just a few lines of scripting within a web page.
- We can import a JavaScript source file into an HTML document; this adds all scripting capabilities to a web page without cluttering the code.

16. What do you understand about cookies?

Fig: Browser cookies

A cookie is generally a small data that is sent from a website and stored on the user's machine by a web browser that was used to access the website. Cookies are used to remember information for later use and also to record the browsing activity on a website.

17. How would you create a cookie?

The simplest way of creating a cookie using JavaScript is as below:

document.cookie = "key1 = value1; key2 = value2; expires = date";

18. How would you read a cookie?

Reading a cookie using JavaScript is also very simple. We can use the document.cookie string that contains the cookies that we just created using that string.

The document.cookie string keeps a list of name-value pairs separated by semicolons, where ‘name’ is the name of the cookie, and ‘value’ is its value. We can also use the split() method to break the cookie value into keys and values.

19. How would you delete a cookie?

To delete a cookie, we can just set an expiration date and time. Specifying the correct path of the cookie that we want to delete is a good practice since some browsers won’t allow the deletion of cookies unless there is a clear path that tells which cookie to delete from the user’s machine.

```
function delete_cookie(name) {
  document.cookie = name + "=; Path=/; Expires=Thu, 01 Jan 1970 00:00:01 GMT;";
}
```

20. What’s the difference between let and var?

Both let and var are used for variable and method declarations in JavaScript. So there isn’t much of a difference between these two besides that while var keyword is scoped by function, the let keyword is scoped by a block.

21. What are Closures in JavaScript?

Closures provide a better, and concise way of writing JavaScript code for the developers and programmers. Closures are created whenever a variable that is defined outside the current scope is accessed within the current scope.

```
function hello(name) {
  var message = "hello " + name;
  return function hello() {
    console.log(message);
  };
}
//generate closure
var helloWorld = hello("World");
//use closure
```

```
helloWorld();
```

22. What are the arrow functions in JavaScript?

Arrow functions are a short and concise way of writing functions in JavaScript. The general syntax of an arrow function is as below:

```
const helloWorld = () => {
  console.log("hello world!");
};
```

23. What are the different ways an HTML element can be accessed in a JavaScript code?

Here are the ways an HTML element can be accessed in a JavaScript code:

- getElementByClass('classname'): Gets all the HTML elements that have the specified classname.
- getElementById('idname'): Gets an HTML element by its ID name.
- getElementbyTagName('tagname'): Gets all the HTML elements that have the specified tagname.
- querySelector(): Takes CSS style selector and returns the first selected HTML element.

24. What are the ways of defining a variable in JavaScript?

There are three ways of defining a variable in JavaScript:

Var

This is used to declare a variable and the value can be changed at a later time within the JavaScript code.

Const

We can also use this to declare/define a variable but the value, as the name implies, is constant throughout the JavaScript program and cannot be modified at a later time.

Let

This mostly implies that the values can be changed at a later time within the JavaScript code.

25. What are Imports and Exports in JavaScript?

Imports and exports help in writing modular code for our JavaScript applications. With the help of imports and exports, we can split a JavaScript code into multiple files in a project. This greatly simplifies the application source code and encourages code readability.

calc.js

```
export const sqrt = Math.sqrt;

export function square(x) {
  return x * x;
}

export function diag(x, y) {
  return sqrt(square(x) + square(y));
}
```

This file exports two functions that calculate the squares and diagonal of the input respectively.

main.js

```
import { square, diag } from "calc";

console.log(square(4)); // 16

console.log(diag(4, 3)); // 5
```

Therefore, here we import those functions and pass input to those functions to calculate square and diagonal.

26. What is the difference between Document and Window in JavaScript?

Document	Window
The document comes under the windows object and can also be considered as its property.	Window in JavaScript is a global object that holds the structure like variables, functions, location, history, etc.

27. What are some of the JavaScript frameworks and their uses?

JavaScript has a collection of many frameworks that aim towards fulfilling the different aspects of the web application development process. Some of the prominent frameworks are:

- React - Frontend development of a web application
- Angular - Frontend development of a web application
- Node - Backend or server-side development of a web application

28. What is the difference between Undefined and Undeclared in JavaScript?

Undefined	Undeclared
Undefined means a variable has been declared but a value has not yet been assigned to that variable.	Variables that are not declared or that do not exist in a program or application.

29. What is the difference between Undefined and Null in JavaScript?

Undefined	Null
Undefined means a variable has been declared but a value has not yet been assigned to that variable.	Null is an assignment value that we can assign to any variable that is meant to contain no value.

30. What is the difference between Session storage and Local storage?

Session storage	Local storage
The data stored in session storage gets expired or deleted when a page session ends.	Websites store some data in local machine to reduce loading time; this data does not get deleted at the end of a browsing session.

32. What is the difference between Event Capturing and Event Bubbling?

Event Capturing	Event Bubbling
This process starts with capturing the event of the outermost element and then propagating it to the innermost element.	This process starts with capturing the event of the innermost element and then propagating it to the outermost element.

33. What is the Strict mode in JavaScript?

Strict mode in JavaScript introduces more stringent error-checking in a JavaScript code.

- While in Strict mode, all variables have to be declared explicitly, values cannot be assigned to a read-only property, etc.
- We can enable strict mode by adding ‘use strict’ at the beginning of a JavaScript code, or within a certain segment of code.

36. What is the difference between Call and Apply?

Call	Apply
In the call() method, arguments are provided individually along with a ‘this’ value.	In the apply() method, arguments are provided in the form of an array along with a ‘this’ value.

40. Can you draw a simple JavaScript DOM (Document Object Model)?