**What is JavaScript**

JavaScript is an object-based scripting language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.

**Application of JavaScript**

JavaScript is used to create interactive websites. It is mainly used for:

* Client-side validation,
* Dynamic drop-down menus,
* Displaying date and time,
* Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
* Displaying clocks etc.

<script>

document.write("Hello JavaScript by JavaScript");

</script>

JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).

2. what is a variable --- answered.

### 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.

Variables are containers for storing data (storing data values).

### 4 Ways to Declare a JavaScript Variable:

* Using var
* Using let
* Using const
* Using nothing
* If you want your code to run in older browsers, you must use var.

In this example, x, y, and z, are variables, declared with the var keyword:

In this example, x, y, and z, are variables, declared with the let keyword:

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

1. <script>
2. function abc(){
3. var x=10;//local variable
4. }
5. </script>
6.  If(10<13){
7.  var y=20;//JavaScript local variable
8.  }

**avaScript global variable**

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

1. <script>
2. var data=200;//gloabal variable
3. function a(){
4. document.writeln(data);
5. }
6. function b(){
7. document.writeln(data);
8. }
9. a();//calling JavaScript function
10. b();

 var value=50;//global variable

 function a(){

 alert(value);

 }

 function b(){

 alert(value);

1.  }

<p>In this example, x, y, and z are undeclared variables.</p>

<p id="demo"></p>

<script>

x = 5;

y = 6;

z = x + y;

document.getElementById("demo").innerHTML =

"The value of z is: " + z;

</script>

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

|  |  |  |
| --- | --- | --- |
| var | let | const |
| There is a global scope as well as a function scope. | There is neither a global scope nor a function scope. | There is neither a global scope nor a function scope. |
| 1. There is no block scope. | There is no block scope. | There is no block scope. |
| It can be reassigned. | cIt cannot be reassigned. | It cannot be reassigned. |

Example 1: Using ‘var’ and ‘let’ variable

var variable1 = 31;

let variable2 = 89;

function catchValues()

{

  console.log(variable1);

  console.log(variable2);

// Both the variables are accessible from anywhere as their declaration is in the global scope

}

window.variable1; // Returns the value 31

window.variable2; // Returns undefined

Example 2: Using ‘const’ variable

const x = {name:"Vijay"};

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

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

const y = 31;

y = 44; // Throws an error

### 60. Rest parameter and spread operator

* Rest Parameter(...)
* Rest parameter is used to declare the function with improved handling of parameters.
* Rest parameter syntax can be used to create functions to perform functions on the variable number of arguments.
* It also helps to convert any number of arguments into an array as well as helps in extracting some or all parts of the arguments.
* Spread Operator(...)
* In a function call, we use the spread operator.
* It's also to spread one or more arguments that are expected in a function call.
* The spread operator is used to take an array or an object and spread them.

### 61. Promises in JavaScript

Promises in JavaScript have four different states. They are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Pending | Fulfilled | Rejected | Settled |
| Pending is an initial state of promise. It is the initial state of promise where it is in the pending state that neither is fulfilled nor rejected. | It is the state where the promise has been fulfilled that assures that the async operation is done. | It is the state where the promise is rejected and the async operation has failed. | It is the state where the promise is rejected or fulfilled. |

Example:

function sumOfThreeElements(...elements)

{

  return new Promise((resolve,reject)=>{

    if(elements.length > 3 )

{

      reject("Just 3 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);

    }

  })

}

### 74. What is Lexical Scoping?

Lexical Scoping in JavaScript can be performed when the internal state of the JavaScript function object consists of the function’s code as well as references concerning the current scope chain.

### 71. Difference between Async/Await and Generators

* Async/Await
* Async-await functions are executed sequentially one after another in an easier way.
* Async/Await function might throw an error when the value is returned.
* Generators
* Generator functions are executed with one output at a time by the generator’s yield by yield.
* The ‘value: X, done: Boolean’ is the output result of the Generator function.

### 88. How do you remove duplicates from a JavaScript array?

There are two ways in which we can remove duplicates from a JavaScript array:

#### By Using the Filter Method

To call the [filter() method](https://www.simplilearn.com/tutorials/javascript-tutorial/arry-filter-in-javascript), three arguments are required. These are namely array, current element, and index of the current element.

#### By Using the For Loop

An empty array is used for storing all the repeating elements.

3. different types of variables keywords and diff b/w them --- answered.

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

### 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. |

* 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

3. console.log in js --- answered.

### 34. What are the scopes of a variable in JavaScript?

The scope of variables in JavaScript is used to determine the accessibility of variables and functions at various parts of one’s code. There are three types of scopes of a variable, global scope, function scope, block scope

* Global Scope: It is used to access the variables and functions from anywhere inside the code.

Example:

var globalVariable = "Hello world";

function sendMessage(){

  return globalVariable; // globalVariable is accessible as it's written in global space

}

function sendMessage2(){

  return sendMessage(); // sendMessage function is accessible as it's written in global space

}

sendMessage2();  // Returns “Hello world”

* Function scope: It is used to declare the function and variables inside the function itself and not outside.

Example:

function awesomeFunction()

{

  var a = 3;

  var multiplyBy3 = function()

{

    console.log(a\*3); // Can access variable "a" since a and multiplyBy3 both are written inside the same function

  }

}

console.log(a); // a is written in local scope and can't be accessed outside and throws a reference error

multiplyBy3(); // MultiplyBy3 is written in local scope thus it throws a reference error

* Block Scope: It uses let and const to declare the variables.

Example:

{

  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

4. different types of operators in js --- answered.

* **Arithmetic Operators**
* **Comparison Operators**
* **Logical Operators**
* **Assignment Operators**
* **Ternary Operators**

**Ternary Operator:**

* **: ? Operator :**   
  It is like the short form of the if-else condition.   
  **Syntax:**

Y = ? A : B

* where A and B are values and if condition is true then Y = A otherwise Y = B.   
  **Example:**

Y = (6>5) ? 6 : 5

therefore Y = 6

* **typeof Operator**

Type of a = number

Type of b = string

Type of c = string

Type of d = object

Type of e = undefined

5. datatypes in js --- answered.

https://www.geeksforgeeks.org/variables-datatypes-javascript/

<https://www.javascripttutorial.net/javascript-data-types/>

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data\_structures

https://www.edureka.co/blog/data-types-in-javascript/

### 7. How do you create an array in JavaScript?

Here is a very simple way of creating [arrays in JavaScript](https://www.simplilearn.com/tutorials/javascript-tutorial/javascript-arrays) using the array literal:

var a = [];

var b = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’];

### 6. How do you create an object in JavaScript?

Since JavaScript is essentially an [object-oriented scripting](https://www.simplilearn.com/tutorials/javascript-tutorial/oop-in-javascript) language, it supports and encourages the [usage of objects](https://www.simplilearn.com/tutorials/javascript-tutorial/javascript-objects) while developing web applications.

const student = {

    name: 'John',

    age: 17

}

6. write a basic function expression and function declaration --- answered.

<https://www.geeksforgeeks.org/difference-between-function-declaration-and-function-expression-in-javascript/>

https://www.sitepoint.com/when-to-use-a-function-expression-vs-function-declaration/

### 14. What is the difference between Function declaration and Function expression?

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

Javascript consists of two data types, primitive data types, and non-primitive data types.

* Primitive Data types: These data types are used to store a single value. Following are the sub-data types in the Primitive data type.
* Boolean Data Types: It stores true and false values.

Example:

var a = 3;

var b =  4;

var c = 3;

(a == b) // returns false

(a == c) //returns true

* Null data Types: It stores either empty or unknown values.

Example:

var z = null;

* Undefined data Types: It stores variables that are only declared, but not defined or initialized.

Example:

var a; // a is undefined

var b = undefined; // we can also set the value of b variable as undefined

* Number Data Types: It stores integer as well as floating-point numbers.

Example:

var x = 4; //without decimal

var y = 5.6; //with decimal

* String data Types: It stores characters and alphanumeric values.

Example:

var str = "Raja Ram Mohan"; //using double quotes

var str2 = 'Raja Rani'; //using single quotes

* Symbols Data Types: It stores unique identifiers for objects.

Example:

var symbol1 = Symbol('symbol');

* BigInt Data Types: It stores the Number data types that are large integers and are above the limitations of number data types.

Example:

var bigInteger =  234567890123456789012345678901234567890;

* Non-Primitive Data Types

Non-Primitive data types are used to store multiple as well as complex values.

Example:

// Collection of data in key-value pairs

var obj1 = {

   x:  43,

   y:  "Hello world!",

   z: function(){

      return this.x;

   }

}

// Data collection with an ordered list

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

|  |  |
| --- | --- |
| 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;  } |

7. brief about arrays --- answered

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array>

<https://www.geeksforgeeks.org/arrays-in-javascript/>

8. if I want to add one element from the back, which method do I use? --- answered.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/add>

<https://www.geeksforgeeks.org/how-to-add-an-object-to-an-array-in-javascript/>

There are 3 popular methods which can be used to insert or add an object to an array.

* **push()**
* **splice()**
* **unshift()**

9. unshift method in an array --- not answered.  
10. the pop method in array --- answered.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/pop>

https://www.geeksforgeeks.org/arrays-in-javascript/?ref=lbp

The **pop()** method removes the **last** element from an array and returns that element. This method changes the length of the array.  
11. difference between == and === ? --- answered.

1. “==” operator is a comparison operator that used to compare the values
2. “===” operator is also a comparison operator that is used to compare the values as well as types.

Example:

var x = 3;

var y = "3";

(x == y)  // it returns true as the value of both x and y is the same

(x === y) // it returns false as the typeof x is "number" and typeof y is "string"

12. tell about null and undefined --- answered.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/null>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined>

13. ternary operators in js --- answered.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator>

14. switch statement in js --- answered.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch>

15. for loop and while loop in js --- answered.

<https://www.tutorialrepublic.com/javascript-tutorial/javascript-loops.php>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/while>

16. tell about the break and continue keywords --- answered.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/break>

17. create a basic object in js ---

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create>

<https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics>

* 1. what is JavaScript?

<https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript>

2. is js a synchronous or async and why? - why not answered

<https://www.freecodecamp.org/news/synchronous-vs-asynchronous-in-javascript/>

<https://www.geeksforgeeks.org/synchronous-and-asynchronous-in-javascript/>

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

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures>

4. in which broader category do string, number, and boolean fall?

<https://www.scaler.com/topics/javascript/pass-by-value-and-pass-by-reference/>

5. other datatypes apart from string, number, and boolean?

6. Explain *Hoisting in javascript*.

<https://developer.mozilla.org/en-US/docs/Glossary/Hoisting>

<https://www.w3schools.com/js/js_hoisting.asp>

<https://www.geeksforgeeks.org/javascript-hoisting/>

7. Difference between “ == “ and “ === “ operators.

## <https://www.scaler.com/topics/javascript/difference-between-double-equals-and-triple-equals-in-javascript/> = Vs == VS === in JavaScript

Here are the important differences between =, ==, and ===

| **=** | **==** | **===** |
| --- | --- | --- |
| = in JavaScript is used for assigning values to a variable. | == in JavaScript is used for comparing two variables, but it ignores the datatype of variable. | === is used for comparing two variables, but this operator also checks datatype and compares two values. |
| It is called as assignment operator | It is called as comparison operator | It is also called as comparison operator |
| The assignment operator can evaluate to the assigned value | Checks the equality of two operands without considering their type. | Compares equality of two operands with their types. |
| It does not return true or false | Return true if the two operands are equal. It will return false if the two operands are not equal. | It returns true only if both values and data types are the same for the two variables. |
| = simply assign one value of variable to another one. | == make type correction based upon values of variables. | === takes type of variable in consideration. |
| == will not compare the value of variables at all. | The == checks for equality only after doing necessary conversations. | If two variable values are not similar, then === will not perform any conversion. |

8. What is hoisted?Variable initialization or variable declarations

<https://www.youtube.com/watch?v=dcAb_gIjqkk>

<https://www.geeksforgeeks.org/javascript-hoisting/>

9. diff b/w var, let, and const

|  |  |  |
| --- | --- | --- |
| var | let | const |
| There is a global scope as well as a function scope. | There is neither a global scope nor a function scope. | There is neither a global scope nor a function scope. |
| 1. There is no block scope. | There is no block scope. | There is no block scope. |
| It can be reassigned. | cIt cannot be reassigned. | It cannot be reassigned. |

Example 1: Using ‘var’ and ‘let’ variable

var variable1 = 31;

let variable2 = 89;

function catchValues()

{

  console.log(variable1);

  console.log(variable2);

// Both the variables are accessible from anywhere as their declaration is in the global scope

}

window.variable1; // Returns the value 31

window.variable2; // Returns undefined

Example 2: Using ‘const’ variable

const x = {name:"Vijay"};

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

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

const y = 31;

y = 44; // Throws an error

#### 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.

10. explain 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 the global scope and functional scope, we are trying to access variables that have not been declared yet. This is called the **Temporal Dead Zone**.

11. 1.console.log(9-“5”)

Uncaught SyntaxError: Invalid or unexpected token

2. console.log(“9”-“5”)

Uncaught SyntaxError: Invalid or unexpected token

3.console.log(1+3+"5")

Ans :=45  
12. what is lexical scope in js?   
13. What are callbacks in js?  
14. why do we use callback?

### 41. Why do we use callbacks?

A callback function is a method that is sent as an input to another function (now let us name this other function "thisFunction"), and it is performed inside the thisFunction after the function has completed execution.

JavaScript is a scripting language that is based on events. Instead of waiting for a reply before continuing, JavaScript will continue to run while monitoring for additional events. Callbacks are a technique of ensuring that a particular code does not run until another code has completed its execution.

15. What are 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"

16. What is Closure?

[Closures](https://www.simplilearn.com/tutorials/javascript-tutorial/javascript-closure) 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();

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

<https://www.tutorialrepublic.com/javascript-tutorial/javascript-get-started.php>

There are 3 ways to include Javascript in HTML:

1. External Javascript, load a Javascript file – <script src="FILE.JS"></script>
2. Internal Javascript, add a block of code in the HTML document itself – <script>DO SOMETHING</script>
3. Inline Javascript, directly add Javascript to an HTML element – <input type="button" value="Test" onclick="FUNCTION()"/>

18. What are the various features introduced in ES2015 in JavaScript?

<https://auth0.com/blog/a-rundown-of-es6-features/>

19. what are map, filter, and reduce?

<https://www.freecodecamp.org/news/javascript-map-reduce-and-filter-explained-with-examples/>

20. What are the parameters and arguments in JavaScript?

<https://wesbos.com/javascript/02-functions/functions-parameters-and-arguments>

21. What are promises in JavaScript?   
22. What is an event loop in JavaScript?

<https://www.geeksforgeeks.org/what-is-an-event-loop-in-javascript/>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop>

23. What are the spread and rest operators in JavaScript?

* Rest Parameter(...)
* Rest parameter is used to declare the function with improved handling of parameters.
* Rest parameter syntax can be used to create functions to perform functions on the variable number of arguments.
* It also helps to convert any number of arguments into an array as well as helps in extracting some or all parts of the arguments.
* Spread Operator(...)
* In a function call, we use the spread operator.
* It's also to spread one or more arguments that are expected in a function call.
* The spread operator is used to take an array or an object and spread them.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax>

<https://www.freecodecamp.org/news/javascript-rest-vs-spread-operators/>

<https://javascript.plainenglish.io/javascript-spread-and-rest-operators-when-to-use-them-1d20f43f4a87>

24. What is callback hell in JavaScript?

### What is a Temporal Dead Zone?

Temporal Dead Zone is a behavior that occurs with variables declared using let and const keywords before they are initialized.

* 1. Give examples of - FlexBox  
     Grid  
     Promises  
     Callbacks

### 23. 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, the first number, the 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, a callback is a function that will be executed after another function gets executed.

### 24. What are the types of errors in javascript?

There are two types of errors in javascript.

1. **Syntax error**: Syntax errors are mistakes or spelling problems in the code that cause the program to not execute at all or to stop running halfway through. Error messages are usually supplied as well.
2. **Logical error**: Reasoning mistakes occur when the syntax is proper but the logic or program is incorrect. The application executes without problems in this case. However, the output findings are inaccurate. These are sometimes more difficult to correct than syntax issues since these applications do not display error signals for logic faults.

### 25. What is memoization?

* 1. Closures

### 20. Explain Closures in JavaScript.

Closures are 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 randomFunc 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.

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

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

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.**

Async Await  
Media Queries

<https://www.geeksforgeeks.org/css-media-queries/> <https://www.w3schools.com/css/css3_mediaqueries.asp>

Prototype  
Generators

* 1. Explain about variables?2. What do you understand by functional and block scope?
* <https://www.geeksforgeeks.org/all-about-functions-and-scopes-in-javascript/>
* 3. How hoisting works with var, let and const?4.Diff. between normal function and arrow function?

Hoisting in javascript is the default process behavior of moving declaration of all the variables and functions on top of the scope where scope can be either local or global.

Example 1:

hoistedFunction();  // " Hi There! " is an output that is declared as function even after it is called

function hoistedFunction(){

  console.log(" Hi There! ");

}

Example 2:

hoistedVariable = 5;

console.log(hoistedVariable); // outputs 5 though the variable is declared after it is initialized

var hoistedVariable;

5. setTimeout vs setInterval?6. Diff. between for of, and for in?  
7. Diff. between forEach and map(array methods)?8. Explain about event loop?9. Explain about closures?  
10. Write a program to remove duplicates number from an array?  
Output based :   
1. console.log(a);  
var a =10;  
Follow up : what will be the output if we change var to let.  
2. function x() {  
var a = 10;  
function y() {  
console.log(a);  
}  
return y;  
}  
x()();  
Follow Up:  
1. What if we move var a =10 after the function y declaration?  
2. What if we change var with let?  
3. console.log("start");  
setTimeout(() => {  
console.log("timeout");  
}, 0);  
Promise.resolve(() => console.log("resolved")).then((res) => res());  
console.log("end");  
  
4. function x() {  
for (var i = 0; i <= 5; i++) {  
setTimeout(() => {  
console.log(i);  
}, i \* 1000);  
}  
}  
x();

### 17. What is currying in JavaScript?

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

Example of a curried function:

function add (a) {

return function(b){

return a + b;

}

}

add(3)(4)

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

### 4. Difference between “ == “ and “ === “ operators.

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

**Example:**

var x = 2;

var y = "2";

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

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

### 5. Difference between var and let keyword in javascript.

Some differences are

1. From the very beginning, the 'var' keyword was used in JavaScript programming **whereas the keyword** 'let' was just added in 2015.
2. The keyword 'Var' has a function scope. Anywhere in the function, the variable specified using var is accessible but in ‘let’ the scope of a variable declared with the 'let' keyword is limited to the block in which it is declared. Let's start with a Block Scope.
3. In ECMAScript 2015, let and const are hoisted but not initialized. Referencing the variable in the block before the variable declaration results in a ReferenceError because the variable is in a "temporal dead zone" from the start of the block until the declaration is processed.

### 6. Explain Implicit Type Coercion in javascript.

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

* **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 is 0(Falsy)

if(y) { console.log(y) } // The code inside this block will run since the value of y is 23 (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.

### 12. 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"

Higher-order functions are the functions that take functions as arguments and return them by operating on other functions

Example:

function higherOrder(fn)

 {

  fn();

}

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

In JavaScript, when a function of an argument is transformed into functions of one or more arguments is called Currying.

Example:

function add (a) {

  return function(b){

    return a + b;

  }

}

add(3)(4)

### 37. 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 the limited functionality of callbacks, 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.

### Difference between Async/Await and Generators usage to achieve the same functionality.

* Generator functions are run by their generator yield by yield which means one output at a time, whereas Async-await functions are executed sequentially one after another.
* Async/await provides a certain use case for Generators easier to execute.
* The output result of the Generator function is always value: X, done: Boolean, but the return value of the Async function is always an assurance or throws an error.

### 49. What are the primitive data types in JavaScript?

A primitive is a data type that isn't composed of other data types. It's only capable of displaying one value at a time. By definition, every primitive is a built-in data type (the compiler must be knowledgeable of them) nevertheless, not all built-in datasets are primitives. In JavaScript, there are 5 different forms of basic data. The following values are available:

1. Boolean
2. Undefined
3. Null
4. Number
5. String

### 50. What is the role of deferred scripts in JavaScript?

The processing of HTML code while the page loads are disabled by nature till the script hasn't halted. Your page will be affected if your network is a bit slow, or if the script is very hefty. When you use Deferred, the script waits for the HTML parser to finish before executing it. This reduces the time it takes for web pages to load, allowing them to appear more quickly.

### 51. What has to be done in order to put Lexical Scoping into practice?

To support lexical scoping, a JavaScript function object's internal state must include not just the function's code but also a reference to the current scope chain.

### 52. What is the purpose of the following JavaScript code?

var scope = "global scope";

function check()

{

var scope = "local scope";

function f()

{

return scope;

}

return f;

}

Every executing function, code block, and script as a whole in JavaScript has a related object known as the Lexical Environment. The preceding code line returns the value in scope

**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 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 the elements array is more than 3, a promise is rejected, or 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

### 38. 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: ${this.grade}, Section:${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}, Section:${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. An error will be thrown if the strict mode rules are not followed.

### 39. What are generator functions?

Introduced in the ES6 version, generator functions are a special class of functions.  
  
**They can be stopped midway and then continue from where they 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 a **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.

### 40. Explain WeakSet in javascript.

In javascript, a 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, that 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

### 41. Why do we use callbacks?

A callback function is a method that is sent as an input to another function (now let us name this other function "thisFunction"), and it is performed inside the thisFunction after the function has completed execution.

JavaScript is a scripting language that is based on events. Instead of waiting for a reply before continuing, JavaScript will continue to run while monitoring for additional events. Callbacks are a technique of ensuring that a particular code does not run until another code has completed its execution.

### 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. |

# Hoisting

<https://developer.mozilla.org/en-US/docs/Glossary/Hoisting>

# JavaScript | console.log() with Examples

<https://www.geeksforgeeks.org/javascript-console-log-with-examples/?ref=leftbar-rightbar>

# What is currying function in JavaScript ?

<https://www.geeksforgeeks.org/what-is-currying-function-in-javascript/>

# Pure Functions in JavaScript

<https://www.geeksforgeeks.org/pure-functions-in-javascript/>

# Understanding Higher-Order Functions in JavaScript

<https://blog.bitsrc.io/understanding-higher-order-functions-in-javascript-75461803bad>

# Closures

https://www.geeksforgeeks.org/closure-in-javascript/

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures>

# Promise

https://www.geeksforgeeks.org/javascript-promises/

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise>

## Nesting Callbacks and Callback Hell

# Callback Functions in JavaScript: Everything You Need to Know

<https://www.javatpoint.com/javascript-callback>

<https://www.geeksforgeeks.org/what-to-understand-callback-and-callback-hell-in-javascript/>

<https://www.javatpoint.com/javascript-callback>

# Difference between block elements and inline elements

<https://www.geeksforgeeks.org/difference-between-block-elements-and-inline-elements/>

<http://web.simmons.edu/~grovesd/comm244/notes/week4/block-inline>

# Difference between CSS Grid and CSS Flexbox

<https://www.geeksforgeeks.org/comparison-between-css-grid-css-flexbox/>

# CSS Box model

<https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model>

<https://www.geeksforgeeks.org/css-box-model/>

var a = 10;

undefined

var b = "25";

undefined

var z1 = a+ 10 +1+b;

undefined

var z2 = b+20+1+b;

undefined

var z3 = 10+10 +a +b;

undefined

console.log(z1)

VM353:1 2125

undefined

console.log(z2)

VM376:1 2520125

undefined

console.log(z3)

VM398:1 3025

Undefined

var array = [18, -42, 21, 6, -50]; array = array.filter(function(x) { return x > -1; }); console.log(array);

VM404:3

1. *(3) [18, 21, 6]*
   1. **0**: 18
   2. **1**: 21
   3. **2**: 6
   4. **length**: 3
   5. [[Prototype]]: Array(0)

<https://www.geeksforgeeks.org/remove-all-negatives-from-the-given-array/>

https://www.geeksforgeeks.org/how-to-find-the-sum-of-all-elements-of-a-given-array-in-javascript/