INTERVIEWBIT

Explain the key features introduced in Javascript ES6 In ES6, JavaScript introduced these key features: Ye Arrow Functionsc M Concise syntax for anonymous functions with lexical scoping. …e Template Literalsc M Enables multiline strings and variable inclusion for improved readability. ²e Destructuring Assignmentc M Simplifies extraction of values from arrays or objects. Ùe Enhanced Object Literalsc M Introduces shorthand notation for defining object methods and dynamic property names. öe Promisesc M Streamlines asynchronous programming with a cleaner, structured approach.

Is javascript a statically typed or a dynamically typed language? JavaScript is a dynamically typed language. In a dynamically typed language, variable types are determined at runtime, allowing a variable to hold values of any type without explicit type declarations. This flexibility can make coding more convenient but may also lead to runtime errors if types are not handled appropriately. JavaScript, being dynamically typed, allows variables to change types during execution and accommodates a wide range of data types without explicit type annotations.

1. map():  Applies a function to each element of an array and creates a new array with the results.8  Example:  const numbers = [1, 2, 3, 4, 5];   const doubledNumbers = numbers.map(number => number \* 2); // [2, 4, 6, 8, 10] ¡a filter():  Creates a new array containing only elements that pass a test implemented by a provided function.8  Example:   const numbers = [1, 2, 3, 4, 5];   const evenNumbers = numbers.filter(number => number % 2 === 0); // [2, 4] MEDIUM www.bosscoderacademy.com 17 reduce(): . Applies a function against an accumulator and each element in an array (from left to right) to reduce it to a single value.A . Example:   const numbers = [1, 2, 3, 4];   const sum = numbers.reduce((accumulator, number) => accumulator + number, 0); // 10

What is difference between Null and Undefined

What is Temporal Dead Zone in Javascript? The Temporal Dead Zone is a phenomenon in JavaScript associated with the use of the let and const keywords, unlike the var keyword. In ECMAScript 6, attempting to access a let or const variable before it is declared within its scope results in a ReferenceError. The term "temporal dead zone" refers to the timeframe during which this occurs, spanning from the creation of the variable's binding to its actual declaration. Let's illustrate this behaviour with an example: Javascript function exampleMethod() { console.log(value1); console.log(value2); var value1 = 1; let value2 = 2; } // Outputs: undefined // Throws a ReferenceError In this example, attempting to access before its declaration causes a ReferenceError due to the temporal dead zone, while accessing results in an output of .

Explain rest parameter in Javascript In JavaScript, the rest parameter is a feature that allows you to represent an indefinite number of arguments as an array. It is denoted by three dots (...) followed by the parameter name. The rest parameter collects all the remaining arguments passed to a function into a single array. Here's a simple example to illustrate the concept: Javascript function sum(...numbers) { return numbers.reduce((total, num) => total + num, 0); } console.log(sum(1, 2, 3, 4, 5)); // Output: 15 In this example, the sum function accepts any of arguments. The rest parameter ... collects all the arguments into an array called . The function then uses the method to sum up all the numbers in the array. It's important to note that the rest parameter must be the last parameter in the function declaration. For example, this is valid:

**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"

### Is javascript a statically typed or a dynamically typed language?

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

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 that is assigned a number type can be converted to a string type:

**var** a = 23;

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

### What do you mean by Self Invoking Functions?

Without being requested, a self-invoking expression is automatically invoked (initiated). If a function expression is followed by (), it will execute automatically. A function declaration cannot be invoked by itself.

Normally, we declare a function and call it, however, anonymous functions may be used to run a function automatically when it is described and will not be called again. And there is no name for these kinds of functions.

**What is the difference between exec () and test () methods in javascript?**

* **test ()** and **exec ()** are RegExp expression methods used in javascript.
* We'll use **exec ()** to search a string for a specific pattern, and if it finds it, it'll return the pattern directly; else, it'll return an 'empty' result.
* We will use a**test ()** to find a string for a specific pattern. It will return the Boolean value 'true' on finding the given text otherwise, it will return 'false'.

### 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.

### 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?

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 the 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.

### 26. 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

### 27. 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- The 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.

**4. 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 the 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 parameters 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 the spread operator is exactly the same as the rest parameter, the spread operator is used to spreading 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 them 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

**7. 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.

**8. 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

**12. 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

**13. Difference between prototypal and classical inheritance**

Programers build objects, which are representations of real-time entities, in traditional OO programming. Classes and objects are the two sorts of abstractions. A class is a generalization of an object, whereas an object is an abstraction of an actual thing. A Vehicle, for example, is a specialization of a Car. As a result, automobiles (class) are descended from vehicles (object).

Classical inheritance differs from prototypal inheritance in that classical inheritance is confined to classes that inherit from those remaining classes, but prototypal inheritance allows any object to be cloned via an object linking method. Despite going into too many specifics, a prototype essentially serves as a template for those other objects, whether they extend the parent object or not.

**7. 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

### 1. What is the output of the following code?

**const** b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

**for** (**let** i = 0; i < 10; i++) {

setTimeout(() => console.log(b[i]), 1000);

}

**for** (**var** i = 0; i < 10; i++) {

setTimeout(() => console.log(b[i]), 1000);

}

**Ans.**

1

2

3

4

5

6

7

8

9

10

undefined

undefined

undefined

undefined

undefined

undefined

undefined

undefined

undefined

undefined

### Conclusion

It is preferable to keep the JavaScript, CSS, and HTML in distinct Separate 'javascript' files. Dividing the code and HTML sections will make them easier to understand and deal with. This strategy is also simpler for several programmers to use at the same time. JavaScript code is simple to update. Numerous pages can utilize the same group of JavaScript Codes. If we utilize External JavaScript scripts and need to alter the code, we must do it just once. So that we may utilize a number and maintain it much more easily. Remember that professional experience and expertise are only one aspect of recruitment. Previous experience and personal skills are both vital in landing (or finding the ideal applicant for the job.

Remember that many JavaScript structured interviews are free and have no one proper answer. Interviewers would like to know why you answered the way you did, not if you remembered the answer. Explain your answer process and be prepared to address it. If you're looking to further enhance your JavaScript skills, consider enrolling in this free JavaScript course by [Scaler Topics](https://www.scaler.com/topics/course/javascript-beginners/" \t "_blank) to gain hands-on experience and improve your problem-solving abilities.

### Recommended Resources

* [JavaScript Cheat Sheet: Basics to Advanced(2023)](https://www.interviewbit.com/javascript-cheat-sheet/)
* [Online Javascript Compiler](https://www.interviewbit.com/online-javascript-compiler/)
* [Top JavaScript Features You Must Know](https://www.interviewbit.com/blog/javascript-features/)
* [50 JavaScript MCQ With Answers](https://www.interviewbit.com/javascript-mcq/)
* [Top 15+ JavaScript Projects for Beginners to Advanced [With Source Code]](https://www.interviewbit.com/blog/javascript-projects/)
* [9 Best JavaScript IDE & Source Code Editors [2023]](https://www.interviewbit.com/blog/javascript-ide/)
* [Top ES6 Interview Questions (2023)](https://www.interviewbit.com/es6-interview-questions/)
* [10 Best JavaScript Books for Beginners to Advanced [2023]](https://www.interviewbit.com/blog/best-javascript-books/)
* [30+ Top Node.js Interview Questions (2023)](https://www.interviewbit.com/node-js-interview-questions/)
* [Typescript vs Javascript: What’s The Difference?](https://www.interviewbit.com/blog/typescript-vs-javascript/)
* [Top 5 JavaScript Libraries You Must Know in 2023](https://www.interviewbit.com/blog/top-javascript-libraries/)
* [Difference Between HTML and JavaScript](https://www.interviewbit.com/blog/difference-between-html-and-javascript/)
* [Javascript Vs jQuery: What’s the Difference? [2023]](https://www.interviewbit.com/blog/javascript-vs-jquery/)
* [Javascript Vs Python: What’s The Difference? [2023]](https://www.interviewbit.com/blog/javascript-vs-python/)
* [Difference Between Java and Javascript](https://www.interviewbit.com/blog/difference-between-java-and-javascript/)
* [Difference between Typescript and Javascript](https://www.interviewbit.com/blog/typescript-vs-javascript/)

### Interview Guides

* [The Ultimate Guide to Acing Your Technical Interview](https://www.interviewbit.com/technical-interview-questions/)
* [300+ Must Do Coding Questions from Interviews](https://www.interviewbit.com/coding-interview-questions/)
* [Mock Interview](https://www.interviewbit.com/mock-interview/)
* [InterviewBit - No.1 Resource for Tech Interview Preparation](https://www.interviewbit.com/blog)

### 2. In JavaScript, how do you turn an Object into an Array []?

**let** obj = { id: "1", name: "user22", age: "26", work: "programmer" };

//Method 1: Convert the keys to Array using - Object.keys()

console.log(Object.keys(obj));

// ["id", "name", "age", "work"]

// Method 2 Converts the Values to Array using - Object.values()

console.log(Object.values(obj));

// ["1", "user22r", "26", "programmer"]

// Method 3 Converts both keys and values using - Object.entries()

console.log(Object.entries(obj));

//[["id", "1"],["name", "user22"],["age", "26"],["work", “programmer"]]

### 3. Write the code to find the vowels

**const** findVowels = str => {

**let** count = 0

**const** vowels = ['a', 'e', 'i', 'o', 'u']

**for**(**let** char **of** str.toLowerCase()) {

**if**(vowels.includes(char)) {

count++

}

}

**return** count

}

### 4. Write the code given If two strings are anagrams of one another, then return true.

**var** firstWord = "Deepak";

**var** secondWord = "Aman";

isAnagram(wordOne, wordTwo); // true

**function** **isAnagram**(one, two) {

//Change both words to lowercase for case insensitivity..

**var** a = one.toLowerCase();

**var** b = two.toLowerCase();

// Sort the strings, then combine the array to a string. Examine the outcomes.

a = a.split("").sort().join("");

b = b.split("").sort().join("");

**return** a === b;

}

### 5. Write the code for dynamically inserting new components.

<html>

<head>

<title>inserting new components dynamically</title>

<script type="text/javascript">

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

var textNode = document.createTextNode(" This is other node");

newP.appendChild(textNode); document.getElementById("parent1").appendChild(newP); }

</script>

</head>

<body> <p id="parent1">firstP<p> </body>

</html>

### 6. Implement a function that returns an updated array with r right rotations on an array of integers a .

**Example:**

Given the following array: **[2,3,4,5,7]**  
Perform **3**right rotations:  
First rotation : [7,2,3,4,5] , Second rotation : [5,7,2,3,4] and, Third rotation: [4,5,7,2,3]

return **[4,5,7,2,3]**

**Answer:**

function rotateRight(arr,rotations){

if(rotations == 0) return arr;

for(let i = 0; i < rotations;i++){

let element = arr.pop();

arr.unshift(element);

}

return arr;

}

rotateRight([2, 3, 4, 5, 7], 3); // Return [4,5,7,2,3]

rotateRight([44, 1, 22, 111], 5); // Returns [111,44,1,22]

### 7. Write a function that performs binary search on a sorted array.

**function** **binarySearch**(arr,value,startPos,endPos){

**if**(startPos > endPos) **return** -1;

**let** middleIndex = Math.floor(startPos+endPos)/2;

**if**(arr[middleIndex] === value) **return** middleIndex;

**elsif**(arr[middleIndex] > value){

**return** binarySearch(arr,value,startPos,middleIndex-1);

}

**else**{

**return** binarySearch(arr,value,middleIndex+1,endPos);

}

}

### 8. Guess the outputs of the following code:

#### **\*\*Note - Code 2 and Code 3 require you to modify the code, instead of guessing the output.**

// Code 1

(**function**(a){

**return** (**function**(){

console.log(a);

a = 23;

})()

})(45);

// Code 2

// Each time bigFunc is called, an array of size 700 is being created,

// Modify the code so that we don't create the same array again and again

**function** **bigFunc**(element){

**let** newArray = **new** Array(700).fill('♥');

**return** newArray[element];

}

console.log(bigFunc(599)); // Array is created

console.log(bigFunc(670)); // Array is created again

// Code 3

// The following code outputs 2 and 2 after waiting for one second

// Modify the code to output 0 and 1 after one second.

**function** **randomFunc**(){

**for**(**var** i = 0; i < 2; i++){

setTimeout(()=> console.log(i),1000);

}

}

randomFunc();

**Answers -**  
  
**Code 1**- Outputs **45**.  
  
Even though a is defined in the outer function, due to closure the inner functions have access to it.  
  
**Code 2**- This code can be modified by using closures,

**function** **bigFunc**(){

**let** newArray = **new** Array(700).fill('♥');

**return** (element) => newArray[element];

}

**let** getElement = bigFunc(); // Array is created only once

getElement(599);

getElement(670);

**Code 3**- Can be modified in two ways:  
  
Using **let**keyword:

**function** **randomFunc**(){

**for**(**let** i = 0; i < 2; i++){

setTimeout(()=> console.log(i),1000);

}

}

randomFunc();

**Using closure:**

**function** **randomFunc**(){

**for**(**var** i = 0; i < 2; i++){

(**function**(i){

setTimeout(()=>console.log(i),1000);

})(i);

}

}

randomFunc();

### 9. Guess the outputs of the following code:

// Code 1

let hero = {

powerLevel: 99,

getPower(){

return this.powerLevel;

}

}

let getPower = hero.getPower;

let hero2 = {powerLevel:42};

console.log(getPower());

console.log(getPower.apply(hero2));

// Code 2

const a = function(){

console.log(this);

const b = {

func1: function(){

console.log(this);

}

}

const c = {

func2: ()=>{

console.log(this);

}

}

b.func1();

c.func2();

}

a();

// Code 3

const b = {

name:"Vivek",

f: function(){

var self = this;

console.log(this.name);

(function(){

console.log(this.name);

console.log(self.name);

})();

}

}

b.f();

Answers:  
  
**Code 1**- Output in the following order:

undefined

42

Reason - The first output is **undefined**since when the function is invoked, it is invoked referencing the global object:

window.getPower() = getPower();

**Code 2**- Outputs in the following order:

global/window object

object "b"

global/window object

Since we are using the arrow function inside **func2, this**keyword refers to the global object.  
  
**Code 3**- Outputs in the following order:

"Vivek"

undefined

"Vivek"

Only in the IIFE inside the function **f**, **this**keyword refers to the global/window object.

### 10. Guess the output of the following code:

**var** x = 23;

(**function**(){

**var** x = 43;

(**function** **random**(){

x++;

console.log(x);

**var** x = 21;

})();

})();

### Answer:

Output is **NaN**.  
  
random() function has functional scope since x is declared and hoisted in the functional scope.  
  
Rewriting the random function will give a better idea about the output:

**function** **random**(){

**var** x; // x is hoisted

x++; // x is not a number since it is not initialized yet

console.log(x); // Outputs NaN

x = 21; // Initialization of x

}

### 11. Guess the outputs of the following code:

// Code 1:

**let** x= {}, y = {name:"Ronny"},z = {name:"John"};

x[y] = {name:"Vivek"};

x[z] = {name:"Akki"};

console.log(x[y]);

// Code 2:

**function** **runFunc**(){

console.log("1" + 1);

console.log("A" - 1);

console.log(2 + "-2" + "2");

console.log("Hello" - "World" + 78);

console.log("Hello"+ "78");

}

runFunc();

// Code 3:

**let** a = 0;

**let** b = false;

console.log((a == b));

console.log((a === b));

**Answers:**  
  
**Code 1**- Output will be **{name: “Akki”}.**  
  
Adding objects as properties of another object should be done carefully.  
  
Writing **x[y] = {name:”Vivek”}**, is same as writing **x[‘object Object’] = {name:”Vivek”}**,  
  
While setting a property of an object, **javascript coerces the parameter into a string.**  
  
Therefore, since **y**is an object, it will be converted to **‘object Object’.**  
  
Both x[y] and x[z] are referencing the same property.  
  
**Code 2**- Outputs in the following order:

11

Nan

2-22

NaN

Hello78

**Code 3**- Output in the following order due to equality coercion:

true

false

### 12. Guess the outputs of the following codes:

// Code 1:

**function** **func1**(){

setTimeout(()=>{

console.log(x);

console.log(y);

},3000);

**var** x = 2;

**let** y = 12;

}

func1();

// Code 2:

**function** **func2**(){

**for**(**var** i = 0; i < 3; i++){

setTimeout(()=> console.log(i),2000);

}

}

func2();

// Code 3:

(**function**(){

setTimeout(()=> console.log(1),2000);

console.log(2);

setTimeout(()=> console.log(3),0);

console.log(4);

})();

**Answers:**

* **Code 1**- Outputs **2**and **12**. Since, even though **let**variables are not hoisted, due to the async nature of javascript, the complete function code runs before the setTimeout function. Therefore, it has access to both x and y.
* **Code 2**- Outputs **3**, three times since variable declared with **var**keyword does not have block scope. Also, inside the for loop, the variable i is incremented first and then checked.
* **Code 3**- Output in the following order:

2

4

3

1 // After two seconds

Even though the second timeout function has a waiting time of zero seconds, the javascript engine always evaluates the setTimeout function using the Web API, and therefore, the complete function executes before the setTimeout function can execute.

GEEKSFORGEEKS

**1. What are the differences between Java and JavaScript?**

JavaScript is a client-side scripting language and Java is object Oriented Programming language. Both of them are totally different from each other.

* [**JavaScript**](https://www.geeksforgeeks.org/javascript-tutorial/)**:** It is a light-weighted programming language (“scripting language”) for developing interactive web pages. It can insert dynamic text into the HTML elements. JavaScript is also known as the browser’s language.
* [**Java**](https://www.geeksforgeeks.org/java/)**:** Java is one of the most popular programming languages. It is an object-oriented programming language and has a virtual machine platform that allows you to create compiled programs that run on nearly every platform. Java promised, “Write Once, Run Anywhere”.

### 7. ****What is negative infinity?****

 The negative infinity is a constant value represents the lowest available value. It means that no other number is lesser than this value. It can be generate using a self-made function or by an arithmetic operation. JavaScript shows the NEGATIVE\_INFINITY value as -Infinity.

### ****Which company developed JavaScript?****

Netscape developed JavaScript and was created by Brenden Eich in the year of 1995.

### ****How to delete property-specific values?****

The [**delete keyword**](https://www.geeksforgeeks.org/javascript-delete-operator/)deletes the whole property and all the values at once like

let gfg={Course: "DSA", Duration:30};

delete gfg.Course;

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

To change the style/class of an element there are two possible ways. We use [document.getElementByID method](https://www.geeksforgeeks.org/html-dom-getelementbyid-method/)

document.getElementById("myText").style.fontSize = "16px;

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

**23. Explain how to read and write a file using JavaScript?**

* The **[readFile()](https://www.geeksforgeeks.org/javascript-program-to-read-text-file/)** functions is used for reading operation.

readFile( Path, Options, Callback)

* The **[writeFile()](https://www.geeksforgeeks.org/javascript-program-to-write-data-in-a-text-file/)** functions is used for writing operation.

writeFile( Path, Data, Callback)

### ****What does**** *****var myArray = [[]];* statement declares?****

In JavaScript, this statement is used to declare a two-dimensional array.

### ****What is the difference between innerHTML & innerText?****

The innerText property sets or returns the text content as plain text of the specified node, and all its descendants whereas the innerHTML property sets or returns the plain text or HTML contents in the elements. Unlike innerText, inner HTML lets you work with HTML rich text and doesn’t automatically encode and decode text.

#### How can you prevent the default behavior of an event in JavaScript?

You can use the preventDefault() method on the event object within an event handler to prevent the default behavior associated with that event.

#### What is the difference between splice() and slice()?

* splice() is used to modify an array by adding, removing, or replacing elements at a specific position.
* slice() is used to create a new array that contains a portion of an existing array, specified by the starting and ending indices.

#### What is the purpose of the fetch() function in JavaScript?

The fetch() function is used to make asynchronous HTTP requests in JavaScript. It returns a Promise that resolves to the response from the server.

Example:

#### JavaScript fetch() code function example.

#### What are the different ways to access an HTML element in JavaScript?

There are three main ways to access an [HTML](https://builtin.com/software-engineering-perspectives/html) element in JavaScript:

1. **Using the getElementById() method:** The getElementById() method takes a string as an argument and returns the HTML element with the specified ID.
2. **Using the getElementsByTagName() method:** The getElementsByTagName() method takes a string as an argument and returns an array of all the HTML elements with the specified tag name.
3. **Using the querySelector() method**: The querySelector() method takes a CSS selector as an argument and returns the first HTML element that matches the selector.