**JAVASCRIPT INTERVIEW QUESTIONS**

1. **What will be output also explain reason**

**A)**

**const value1 = “5”;**

**const value2 = 9**

**Sum = value1 + value2**

**console.log(sum);**

**Answer:**

**The output will be "59" because When you use the + operator with strings in JavaScript, it concatenates them instead of performing numeric addition.**

**B)**

**const value1 = “5”;**

**const value2 = 9;**

**let sum = Number(value1) + value2;**

**console.log(sum);**

**Answer:**

**The output will be 14. By using Number(value1), you explicitly convert the string "5" to a number,**

**C)**

**console.log(![])**

**The output will be false. When used with an array [], it first converts the array to a boolean. An empty array is considered true so the logical NOT of [] is false**

**console.log([] == ![])**

**The output will be true , This is because ![] evaluates to false, and when comparing an empty array [] to false, type coercion converts [] to an empty string, and both sides are considered equal.**

**console.log([] === ![])**

**The output will be true. This is because the empty array [] is considered true in JavaScript, and the ![] (not empty array) is false, the comparison between [] and false results in true.**

**console.log(null == false )**

**The output will be false. In JavaScript, when using loose equality (==), null is only equal to undefined, and not equal to any other value.**

**console.log(null == true )**

**The output will be false , This is because null is not equal to true in terms of value or type.**

**D)**

**console.log(-"giddyup" + 409)**

**The output will be NaN (Not a Number). The unary negation operator (-) attempts to convert the string "giddyup" to a number, but since the string cannot be directly converted to a numeric value, it results in NaN. Adding 409 to NaN still results in NaN.**

1. **Predict the output with type**

**let myVar\_1 = 10 + ‘ ‘;**

**console.log(myVar\_1);**

**The value of myVar\_1 will be the string "10 ".**

**(The space is treated as a string, leading to concatenation)**

**let myVar\_2 = 10.5 + ' ';**

**console.log(myVar\_2);**

**The output will be the string '10.5 '**

**(The number is implicitly converted to a string during concatenation with the space.)**

**let myVar\_3 = -10.5 + ' ';**

**console.log(myVar\_3);**

**The output will be the string '-10.5 '**

**(The number is implicitly converted to a string during concatenation with the space.)**

**let myVar\_4 = true+ ' ';**

**console.log(myVar\_4);**

**The output will be the string 'true '**

**(The boolean value is implicitly converted to a string during concatenation with the space)**

1. **Explain difference between “==” and “===” operators**

**Answer:**

**Loose Equality (==):**

**\* Performs type coercion, meaning it attempts to convert the operands to the same type before making the comparison.**

**\* For example, 1 == '1' evaluates to true because the string is coerced to a number for the comparison.**

**Strict Equality (===):**

**\* Does not perform type coercion, and the comparison is made without converting the operands.**

**\* For example, 1 === '1' evaluates to false because the operands are of different types.**

1. **Difference between var and let keyword in javaScript**

**Answer:**

**\* var is function-scoped. It means the variable is only accessible within the function where it is declared.**

**\* var allows for variable redeclaration within the same scope without an error.**

**\* let is block-scoped. It is limited to the block, statement, or expression where it is defined.**

**\* let does not allow redeclaration of the same variable within the same block scope.**

1. **Explain Implicit Type Coercion in javascript with example**

**Answer:**

**Implicit type coercion in JavaScript refers to the automatic conversion of data types during certain operations without explicit instruction from the programmer.**

**Example:**

**let num = 5; // num is a number**

**let str = "10"; // str is a string**

**let result = num + str;**

**console.log(result); // Outputs "510"**

1. **Is javascript a statically typed dynamically typed language?**

**Answer:**

**JavaScript is considered a dynamically typed language. In dynamically typed languages, variable types are determined at runtime, and you don't need to explicitly declare the data type of a variable when you define it**

1. **What is NaN property in JavaScript?**

**Answer:**

**In JavaScript, NaN stands for "Not a Number." It is a special value that represents the result of an operation that should return a number, but the actual result is not a valid number**

1. **Explain passed by value and passed by reference**

**Answer:**

**Passed by Value:**

**\* Iwhen you pass a variable to a function, a copy of the actual value is passed.**

**\* Changes made to the parameter inside the function do not affect the original variable outside the function.**

**Passed by Reference:**

**\* when you pass a variable to a function, you are passing a reference to the original variable's memory location.**

**\* Changes made to the parameter inside the function affect the original variable outside the function.**

1. **What is an Immediately Invoked Function in JavaScript?**

**Answer:**

**An Immediately Invoked Function Expression (IIFE) in JavaScript is a function that is defined and executed immediately after its creation.**

1. **What do you mean by strict mode in javascript and characteristics of javascript strict-mode?**

**Answer:**

**Strict mode in JavaScript is a feature that was introduced to enforce better coding practices and catch common coding mistakes.**

**Some characteristics of JavaScript strict mode include:**

**\* Variable Declaration:**

**\* Global Object:**

**\* Assignment Restrictions:**

**\* Octal Literal Syntax:**

**\* Duplicate Parameter Names**

1. **Explain Higher Order Functions in javascript**

**Answer:**

**Higher Order Functions (HOF) in JavaScript are functions that can take other functions as arguments or return functions as their results.**

1. **Explain "this" keyword.**

**Answer:**

**In JavaScript, the this keyword is a reference to the current execution context or the object to which a function belongs. The value of this depends on how a function is invoked.**

1. **Predict the output**

**A)**

**function doSomething(){**

**console.log(this);**

**}**

**doSomething();**

**Answer:**

**Window**

**B)**

**var obj = {**

**name : "vivek",**

**getName: function(){**

**console.log(this.name);**

**}**

**}**

**obj.getName();**

**Answer:**

**vivek**

**C)**

**var obj = {**

**name : "vivek",**

**getName: function(){**

**console.log(this.name);**

**}**

**}**

**var getName = obj.getName;**

**var obj2 = {name:"akshay",getName};**

**obj2.getName();**

**Answer:**

**akshay**

**D)**

**var obj1 ={**

**address : "Mumbai,India",**

**getAddress: function(){**

**console.log(this.address);**

**}**

**}**

**var getAddress = obj1.getAddress;**

**var obj2 = {name:"akshay"};**

**obj2.getAddress();**

**Answer:**

**TypeError: obj2.getAddress is not a function**

1. **Explain call(), apply() and bind() methods**

**Answer:**

1. **call => We use call when we want to borrow a method only once with a defined number of parameters.**
2. **apply => We use apply when we want to borrow a method only once with n number of parameters.**
3. **bind => We use bind when we want to use a method multiple times and we want to make a permanent copy of that method.**
4. **Implement your own call() , bind() and apply()**

**Answer:**

1. **Ownbind()**

**let cap = {**

**name: "Steve",**

**team: "cap",**

**sayHi: function (mem1, mem2) {**

**console.log(`${this.name} ${this.team}`);**

**console.log(" Say hi", mem1, mem2);**

**},**

**};**

**let ironMan = {**

**name: "Tony",**

**team: "iron man",**

**};**

**Function.prototype.myBind = function myBind(args1) {**

**var fn = this // fn() sayHi**

**return function(...args) {**

**fn.apply(args1, args)**

**}**

**}**

**var bounded = cap.sayHi.myBind(ironMan)**

**bounded('member1', 'member2')**

1. **Owncall()**

**function greet(age, country) {**

**console.log(`${this.name}, ${age}, ${country}`)**

**}**

**var obj = {**

**name : 'Athul',**

**}**

**Function.prototype.myCall = function (context, ...args ) {**

**context.requiredFn = this**

**context.requiredFn(...args)**

**delete context.requiredFn**

**}**

**greet.myCall(obj, '24', 'USA')**

1. **Ownapply()**

**function greet(age, country) {**

**console.log(`${this.name}, ${age}, ${country}`)**

**}**

**var obj = {**

**name : 'Athul',**

**}**

**Function.prototype.myCall = function (context, ...args ) {**

**context.requiredFn = this**

**context.requiredFn(...args)**

**delete context.requiredFn**

**}**

**greet.myCall(obj, ['24', 'USA'])**

1. **What is currying in JavaScript?**

**Answer:**

**Currying in JavaScript is a technique where a function with multiple arguments is transformed into a sequence of functions, each taking a single argument.**

**Example:**

**function add(x) {**

**return function(y) {**

**return x + y;**

**};**

**}**

**const addCurried = add(5);**

**const result = addCurried(3);**

**console.log(result); // Output: 8**

1. **Explain scope and scope chain in javaScript with examples**

**Answer:**

**Local Scope:**

**Variables declared inside a function have local scope, meaning they are only accessible within that function.**

**Example:**

**function exampleFunction() {**

**var localVar = "I am local!";**

**console.log(localVar); // Accessible within the function**

**}**

**exampleFunction();**

**console.log(localVar); // Error: localVar is not defined outside the function**

**Global Scope:**

**Variables declared outside any function or block have global scope, meaning they are accessible throughout the entire program.**

**Example:**

**var globalVar = "I am global!";**

**function exampleFunction() {**

**console.log(globalVar); // Accessible within the function**

**}**

**exampleFunction();**

**console.log(globalVar); // Accessible outside the function**

**Scope chain:**

**The scope chain in JavaScript refers to the hierarchy of scopes that the JavaScript engine traverses to resolve the value of a variable. When a variable is accessed, the JavaScript engine looks for the variable in the current scope and then moves up the scope chain until it finds the variable or reaches the global scope**

**Example:**

**var globalVar = "I am global!";**

**function outerFunction() {**

**var outerVar = "I am outer!";**

**function innerFunction() {**

**var innerVar = "I am inner!";**

**console.log(innerVar); // Access innerVar**

**console.log(outerVar); // Access outerVar from the outer scope**

**console.log(globalVar); // Access globalVar from the global scope**

**}**

**innerFunction();**

**// console.log(innerVar); // Error: innerVar is not defined in this scope**

**}**

**outerFunction();**

**// console.log(outerVar); // Error: outerVar is not defined in this scope**

**// console.log(innerVar); // Error: innerVar is not defined in this scope**

**console.log(globalVar); // Access globalVar from the global scope**

1. **Explain Closures in JavaScript.**

**In JavaScript, a closure is created when a function is defined within another function, allowing the inner function to access variables from the outer function even after the outer function has finished executing**

1. **What are object prototypes in javascript**

**Answer:**

**object prototypes are used to enable inheritance and property sharing between objects. Each object in JavaScript is linked to a prototype object from which it can inherit properties.**

**When you access a property or method on an object, JavaScript first looks for that property or method within the object itself. If it doesn't find it, it searches the object's prototype, and this process continues up the prototype chain until it reaches the top-level object, often Object.prototype.**

1. **Predict the output:**

**A)**

**function real(){**

**console.log(" i am real. alwways run me");**

**}**

**function real(){**

**console.log("no i am real one");**

**}**

**real();**

**function real(){**

**console.log("you both are wasted");**

**}**

**Answer:**

**You both are wasted**

**B)**

**var b =1;**

**function outer(){**

**var b =2;**

**function inner(){**

**b++;**

**console.log(b);**

**var b = 3;**

**console.log(b);**

**}**

**inner();**

**}**

**outer();**

**Answer:**

**NaN**

**3**

**c)**

**var d =1;**

**function test(){**

**d = "2";**

**console.log(typeof d);**

**function d(){};**

**};**

**test();**

**console.log(typeof d);**

**Answer:**

**string**

**number**