ADVANCE JAVASCRIPT

**Module-1: Node – JavaScript Fundamental**

1. **What is the difference between Java & JavaScript?**

**Ans:**

Java and JavaScript are two distinct programming languages with different purposes and characteristics:

**1. Origin and History:**

**- Java:** Java was developed by Sun Microsystems (now owned by Oracle) in the mid-1990s. It's known for its portability and is often used for building desktop applications, web applications (using Java Servlets or JSP), and Android mobile apps.

**- JavaScript:** JavaScript was also created in the mid-1990s but by Netscape Communications. It was initially designed for web development to add interactivity to web pages.

**2. Type:**

**- Java:** Java is a statically typed language, which means that variable types are explicitly declared and checked at compile-time.

**- JavaScript:** JavaScript is a dynamically typed language, which means that variable types are determined at runtime.

**3. Usage:**

**- Java:** Java is commonly used for building server-side applications, Android apps, and large-scale enterprise applications.

**- JavaScript:** JavaScript is primarily used for front-end web development, where it runs in web browsers to make web pages interactive. It's also used on the server-side (Node.js) and for game development (e.g., with libraries like Phase).

**4. Syntax:**

**- Java:** Java's syntax is similar to C++ and follows a more traditional programming style with classes and objects.

**- JavaScript:** JavaScript's syntax is influenced by C and is more forgiving and flexible. It uses a prototype-based object model.

**5. Execution Environment:**

**- Java:** Java code is compiled into byte code, which is executed on the Java Virtual Machine (JVM).

**- JavaScript:** JavaScript code is executed by web browsers (client-side) or on server-side platforms like Node.js.

**6. Typical Use Cases:**

**- Java:** Enterprise applications, Android app development, server-side applications, and scientific computing.

**- JavaScript:** Web development (front-end and back-end), browser scripting, and creating web-based games.

**7. Relation:** Despite the similar names, Java and JavaScript are not directly related. They have different syntax, use cases, and purposes. The similarity in names is largely a historical artifact.

1. **What is JavaScript?**

**Ans:** JavaScript is a widely-used programming language primarily used for web development. It allows developers to add interactivity, manipulate web page content, and create dynamic web applications. JavaScript is commonly used alongside HTML and CSS to create interactive features like forms, animations, and responsive web designs. It can be executed in web browsers, making it an essential technology for front-end web development, but it can also be used on the server-side with environments like Node.js. JavaScript is known for its versatility and is a fundamental part of modern web development.

1. **What are the data types supported by JavaScript?**

**Ans:** JavaScript supports several data types, including:

**1. Number:** represents both integer and floating-point numbers.

**2. String:** Represents text and is enclosed in single (' '), double (" "), or back ticks (\` \`).

**3. Boolean:** Represents true or false values.

**4. Null:** represents the intentional absence of any object value.

**5. Undefined:** represents an uninitialized or undeclared variable.

**6. Object:** a complex data type that can store key-value pairs. Objects can be created using curly braces {}.

**7. Array:** a special type of object used to store a collection of values, accessed by index.

**8. Function:** a callable object that can execute a block of code.

**9. Symbol:** Introduced in ECMAScript 6 (ES6), symbols are unique and can be used as object property keys.

**10. Big Int:** Introduced in ES11 (ES2020), BigInt allows you to work with arbitrarily large integers.

1. **What are the scopes of a variable in JavaScript?**

**Ans:** In JavaScript, the scope of a variable refers to where in your code that variable can be accessed or modified. There are two main types of variable scope:

1**. Global Scope:** Variables declared outside of any function or block have global scope. They can be accessed and modified from anywhere in your JavaScript code. Global variables are defined using the `var` keyword (or `let` and `const` if declared at the top level of a JavaScript file or script).

**Example:**

javascript

var globalVariable = 10;

function myFunction() {

console.log(globalVariable); // Accessible inside the function

}

2. **Local Scope**: Variables declared inside a function or block have local scope. They are only accessible within that function or block. This helps prevent naming conflicts and keeps variables isolated to specific parts of your code. Local variables are defined using `var`, `let`, or `const` within a function or block.

**Example:**

javascript

function myFunction() {

var localVariable = 5; // Local variable

console.log(localVariable); // Accessible here

}

console.log(localVariable); // Error, not accessible outside the function

In addition to global and local scope, JavaScript also has function-level scope (variables declared with `var` are function-scoped), and block-level scope (variables declared with `let` and `const` are block-scoped).

Keep in mind that modern JavaScript coding practices tend to favor using `let` and `const` over `var` due to their more predictable scoping rules and better support for block-level scope.

1. **What is callback?**

**Ans:**  callback is a function or piece of code that is passed as an argument to another function. It allows the receiving function to call back or execute the provided code at a specific time or when a certain condition is met. Callbacks are commonly used in programming, especially in event-driven and asynchronous scenarios, to handle tasks like responding to user input, handling network requests, or managing asynchronous operations.

For example, in JavaScript, you might use a callback function to specify what should happen after a file is successfully loaded from a server or when a button is clicked. This allows you to define custom behavior that occurs when an event or task is completed. Callbacks are fundamental in achieving non-blocking behavior and enabling event-driven programming in many programming languages.

1. **What is Closure? Give an example**.

**Ans:** In programming, a closure is a function that retains access to variables from its containing or outer scope, even after that outer scope has finished executing. Closures are often used to create functions with persistent state or to encapsulate functionality.

**Here's a simple JavaScript example:**

javascript

function outerFunction() {

let outerVariable = 10;

function innerFunction() {

console.log(outerVariable); // innerFunction can still access outerVariable

}

return innerFunction;

}

const closureFunc = outerFunction();

closureFunc(); // This will log "10" because innerFunction is a closure that retains access to outerVariable

In this example, `innerFunction` is a closure because it can access the `outerVariable` even after `outerFunction` has completed execution. When `closureFunc` is called, it still has access to `outerVariable` and logs its value, which is "10."

1. **What is the difference between the operators ‘==’ & ‘===’?**

**Ans:** The operators '==' and '===' are used for comparison in programming, but they have different behaviors:

**1. '==':** This is the equality operator and is used for loose or type-coercing equality comparison. It compares the values on both sides of the operator and returns true if they are equal after type coercion. For example, '1' == 1 would be true because the string '1' is coerced into a number before comparison.

**2. '===':** This is the strict equality operator and is used for strict or type-safe equality comparison. It not only compares the values but also checks that the data types are the same. It returns true only if both the values and their data types are identical. For example, '1' === 1 would be false because the data types (string and number) are different.

In general, it's recommended to use '===' for most comparisons because it avoids unexpected type coercion and ensures a more precise comparison. '==' should be used with caution, as it can lead to unexpected results if you're not aware of how type coercion works in your programming language.

1. **What is the difference between null & undefined?**

**Ans:** In JavaScript, `null` and `undefined` are both special values, but they have different meanings:

**1. `null`:** It is a deliberate assignment value that represents the absence of any object value or a reference to a non-existent or unknown object. In other words, when you set a variable or property to `null`, you are explicitly saying that it has no value or should be empty.

**2. `undefined`:** It typically indicates that a variable or property has been declared but has not been assigned a value yet. It's the default value for function parameters that haven't been provided, and it can also indicate a missing property in an object.

**Here's a simple example to illustrate the difference:**

javascript

let a = null; // 'a' is explicitly set to null

let b; // 'b' is declared but not assigned, so it's undefined

console.log(a); // Output: null

console.log(b); // Output: undefined

In summary, `null` is a value you assign to indicate intentional absence of a value, while `undefined` typically represents the absence of an assigned value or a missing property.

1. **What would be the result of 2+5+”3”?**

**Ans:** The result of the expression 2 + 5 + "3" would be the string "73". In this expression, the numbers 2 and 5 are added together to get 7, and then the string "3" is concatenated to the result, forming the string "73".

1. **What is the difference between Call and Apply?**

**Ans:** `call()` and `apply()` are both methods in JavaScript that are used to invoke functions, but they differ in how they pass arguments to the function being called.

**1. `call()`:**

- The `call()` method is used to call a function with a given `this` value and individual arguments passed as separate arguments.

- You provide the function's `this` value as the first argument, and then you can pass any number of additional arguments as individual arguments.

**- Example:**

javascript

function greet(name) {

console.log(`Hello, ${name}!`);

}

greet.call(null, 'John');

// Output: Hello, John!

**2. `apply()`:**

- The `apply()` method is also used to call a function with a given `this` value, but it accepts arguments as an array or an array-like object.

- You provide the function's `this` value as the first argument, and the second argument is an array (or array-like object) containing the arguments.

**- Example:**

javascript

function greet(name, age) {

console.log(`Hello, ${name}! You are ${age} years old.`);

}

greet.apply(null, ['John', 30]);

// Output: Hello, John! You are 30 years old.

In summary, `call()` and `apply()` are similar in that they allow you to specify the `this` value for a function, but they differ in how they pass arguments. `call()` uses individual arguments, while `apply()` uses an array of arguments. This distinction can be useful in different situations depending on how your function is designed to accept arguments.