Article 9 – JavaScript

**Data Types**Variables in JavaScript can be declared using the keywords ‘let’ and ‘const’. The keyword ‘var’ is now obsolete.

Defines the kind of data a variable can hold. They can be further classified into two categories: primitive and non-primitive. Primitive data types are immutable and represent a single value. They are stored by value (stored directly). Non-primitive data types are objects that are stored by reference.

Primitive Data Types: String, Number, Boolean, null

Non-Primitive/Object Data Types: Object, Array, Function

The “typeof” function is used to check the type of any primitive data type.

The “instanceof” function is used to check the type of a non-primitive data type.

**Scope of a Data Type**

Refers to where the variable to accessible in the code. It is essential for understanding code and avoiding bugs.

1) Global Scope

Variables declared outside any function. They can be accessed anywhere in the code.

2) Function Scope

Variables declared inside a function. Only accessible within that function.

3) Lexical Scope

Inner functions can access variables of their outer functions.

Scope Chaining: If a variable isn’t found in its current scope, the program looks outward through the scope chain.

var v/s let v/s const:

(i) var

Function-scoped – variables defined with a “var” are function-scoped.   
If declared outside a function, it’s globally scoped.  
Not block-scoped – ignores the {} blocks like in loops and conditionals

if (true) {

var a = 5;

}

console.log(a); // Accessible (not block-scoped)

(ii) let

Block-scoped – only accessible within the {} block it’s declared in, like if-else and conditionals.  
Cannot be redeclared within the same scope.

(iii) const

Same block-scope behavior as ‘let’.  
However, the value cannot be reassigned again within the same block (unlike let).

if (true) {

const PI = 3.14;

console.log(PI); // Accessible

}

console.log(PI); // ReferenceError

**Operators**

Symbols or keywords used to perform operations on values and variables.

Arithmetic Operators: Used to perform mathematical operations.

E.g. %, ++, --

Assignment Operators: Used to assign values to variables

E.g. =, +=, -=, \*=. /=, %=

Comparison Operators: Used to compare values and return a Boolean

E.g. ==, ===, !=, !==, >, <

Logical Operators: Used to compare Boolean values

E.g, ||, &&, !

String Operators: Used to concatenate strings

E.g. +, +=

Type Operators: Used to determine the data type of a variable

typeof 🡪 Returns the type of the variable

instanceof 🡪 Returns the instance of an object

Ternary Operator: Short-hand for an if-else statement

condition ? valueIfTrue : valueIfFalse

**Control Flow**

By default, the code executes linearly from top to bottom, one line at a time. However, control flow statements allow one to alter the flow of code based on conditions.

1) Conditionals

(i) if-else

let age = 20;

if (age < 18) {

console.log("Minor");

} else if (age >= 18 && age < 60) {

console.log("Adult");

} else {

console.log("Senior");

}

(ii) switch

let fruit = "apple";

switch (fruit) {

case "apple":

console.log("Apples are red");

break;

case "banana":

console.log("Bananas are yellow");

break;

default:

console.log("Unknown fruit");

}

2) Loops

(i) for

for (let i = 0; i < 3; i++) {

console.log(i); // 0, 1, 2

}

(ii) while

let i = 0;

while (i < 3) {

console.log(i);

i++;

}

(ii) do-while

let i = 0;

do {

console.log(i);

i++;

} while (i < 3);

3) Loop Control

break: exits the loop in execution

continue: skips the current iteration and continues to the next