FACULTY NAME:J.VIJAY SATHIARAJ
DESIGNAION:GUEST LECTURER
DEPARTMENT:GEOGRAPHY
CLASS:M.TECH
SEMESTER:II
SUBJECT:JAVA SCRIPTING
SUBJECT CODE:VAC-1

Syntax, Variables, Values, Data Types - Data Types - Expressions and Operators - Control structures

INTRODUCTION

JavaScript (js) is a light-weight object-oriented programming 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. JavaScript was originally developed as live script by Netscape in 1990’s.javascript was created by Brendan Eich in 1995 during his time at Netscape communication. Since then, it has been adopted by all other graphical web browsers.

Although, JavaScript has no connectivity with Java programming language. The name was suggested and provided in the times when Java was gaining popularity in the market. In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language

FUTURE OF JAVASCRIPT

1. Light Weight Scripting Language

JavaScript is a lightweight scripting language because it is made for data handling at the browser only. Since it is not a general-purpose language so it has a limited set of libraries. Also as it is only meant for client-side execution and that too for web applications, hence the lightweight nature of JavaScript is a great feature.

2. Dynamic Typing

JavaScript supports dynamic typing which means types of the variable are defined based on the stored value. For example, if you declare a variable x then you can store either a string or a Number type value or an array or an object. This is known as dynamic typing.

3.Object-Oriented Programming Support

Starting from ES6, the concept of class and OOPs has been more refined. Also, in JavaScript, two important principles with OOP in JavaScript are Object Creation patterns (**Encapsulation**) and Code Reuse patterns (**Inheritance**). Although JavaScript developers rarely use this feature but its there for everyone to explore.

4. Platform Independent

This implies that JavaScript is platform-independent or we can say it is portable; which simply means that you can simply write the script once and run it anywhere and anytime. In general, you can write your JavaScript applications and run them on any platform or any browser without affecting the output of the Script.

5. Prototype-based Language

JavaScript is a prototype-based scripting Language. This means java script uses prototypes instead of classes or inheritance. In languages like Java, we create a class and then we create objects for those classes. But in JavaScript, we define object prototype and then more objects can be created using this object prototype.

6**. Interpreted Language**

JavaScript is an interpreted language which means the script written inside javascript is processed line by line. These Scripts are interpreted by JavaScript interpreter which is a built-in component of the Web browser.

7. **Client-side Validations**

This is a feature which is available in JavaScript since forever and is still widely used because every website has a form in which users enter values, and to make sure that users enter the correct value, we must put proper validations in place.

JavaScript Syntax

JavaScript has its own syntax and programming style. Syntax of a language **defines rules of writing the code** in that language.
JavaScript uses its own syntax to create variable, function, statements, etc. but if you have knowledge of any programming language like C, C++ or Java, it won't be very difficult for you to understand the syntax of JavaScript.

1. A semicolon at the end of every statement (is Optional)
Semicolons are used to **terminate the JavaScript code statements**. A code statement is a line of code and we can mark the end of each code line by adding a **semicolon**. In languages like C, C++, Java, etc. we use a semicolon at the end of each code statement.

In JavaScript, statements are generally followed by a semicolon(;) which indicates termination of the statement.
However, it is **not necessary to use a semicolon**(;) at the end of the statement.
The following example shows two valid declarations of a JavaScript statement:

```
var var_name = value;
var var_name
```

In this example, the first statement is terminated using a semicolon, while the second statement is terminated without any semicolon. Both statements are valid in Java script.

2.JavaScript White Spaces and Line Breaks

JavaScript interpreter **ignores the tabs, spaces, and newlines** that appear in the script, except for strings and regular expressions.

Below we have a simple code example where we have added an extra line break after each code statement and then the code without any space at all. Both are acceptable in JavaScript.

```
var i =  10;
var j =  20;
var a = i+j;
var b = i-j
```

You can write a more clean and readable JavaScript code by using multiple spaces as JavaScript ignores multiple spaces.

3. JavaScript Case Sensitivity

JavaScript is a **case sensitive** language, which means all the keywords, function names, variable names or identifiers should be typed with the consistent casing of letters. In general, a literal should start with a small letter and to combine the long name, we can use the camel case style too.

Let's take an example, in the code below, all the three variables are distinct from each other because the first one is in the lower case, the second one starts with a capital letter, and the third one is in the upper case.

```
var marks;
 var Marks;
var MARKS;
```

4. JavaScript Comments

Comments refer to the **text or code in a program that is ignored at the time of executing the program**. Comments are **used to provide additional information** in the code, such as the description of code. And it is considered as **good practice to add comments** to your code.

Similar to other programming languages like C, C++, etc., JavaScript also defines two types of comments.

•Single line Comments
•Multiline Comments

5.JavaScript Statements

Statements are known as the **instructions in any programing language**, these statements are set to be executed by the computer as a computer program.

JavaScript code is nothing but a list of these programming statements, these statements are collections of values, operators, expressions, keywords, and comments.

```
<html>
 <head>
 <title>JavaScript Statement
</title>
</head>
 <body>
<script>
document.write("this is a text") ; // JavaScript statement
 var a = 10+20; // Another Statement
```

```
document. Write(a);
</script>
 </body>
 </html>
```

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container. Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

```
<script type="text/javascript">
<!--
var money;
var name;
```

You can also declare multiple variables with the same var keyword as follows:

```
<script type="text/javascript">
<!--
var money, name;
</script>
```

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.
For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
<script type="text/javascript">
<!--
var name = "Ali";
var money;
money = 2000.50;
//-->
</script>
```

Note: Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice. JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript
during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

1) Global Variables: A global variable has global scope which means it can be defined anywhere in your JavaScript code.

2)Local Variables: A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind

1. You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, break or Boolean variable names are not valid.

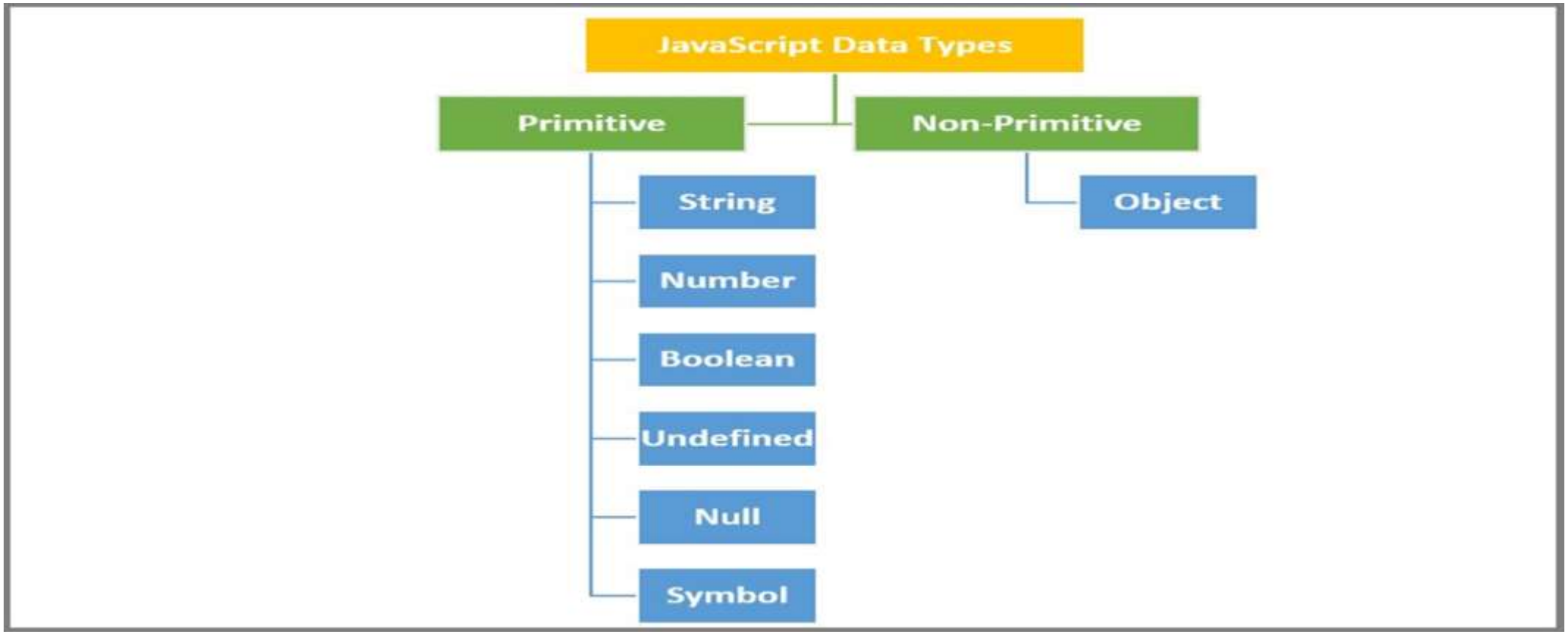
2. JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, 123test is an invalid variable name but _123test is a valid one.

3. JavaScript variable names are case-sensitive. For example, Name and name are two different variables.

Data Types in JavaScript

What are Data Types in JavaScript?

Data types in JavaScript define the data type that a variable can store. JavaScript includes primitive and non-primitive data types. The primitive data types in JavaScript include string, number, boolean, undefined, null, and symbol. The non-primitive data type includes the object. A variable of primitive data type can contain only a single value.



JavaScript Data Types

Primitive Data Types

These are the lowest level of data value in the JavaScript programming language. They define immutable values – which can't be changed.

String

A string data type is a group of characters or textual content enclosed by single quotes (‘ ‘) or double-quotes (” “), or tick signs (` `).

For Example

var str1 = 'Hello, World!'; *// Output: Hello, World!*

* var str2 = "His name is ‘John’"; // Output: His name is ‘John’ *

Number

A number data type is a numeric value. It can be an integer (positive or negative), floating-point, or exponential.

For Example

var a=47; // positive integer value

var b=629.5; // floating point

var c=-5; // number with negative value

The number data type also includes special values such as Infinity, -Infinity, and Nan (not a number).
Infinity – when a positive number is divided by zero, the result is Infinity
-Infinity – when a negative number is divided by zero, the result is -Infinity
For Example

```
var x = 25;
var y = -25;
var z = 0;
alert(x/z);  // returns Infinity
alert(y/z);  // returns -Infinity
```

Nan – when we perform any operation between a number and a non-numeric value, the result is Nan.

Boolean

Boolean is a logical data type used to compare two variables or check a condition. It has only two values, namely, true and false.
For Example – Comparing two variables

```
var a =10;
var b=9;
a==b // returns false
```
For Example – Checking a condition
```
If(x<y){
alert(x is a smaller number than y);
}
```
Undefined

An undefined data type means that a variable is declared and not assigned any value. It is the default value of a variable that has not been defined.

For Example
```
var x;   // undefined
console.log(x); // returns undefined
```
Null

A null value means no value. It means an empty value or absence of value.

For Example
```
var x = null;
console.log(x);   // returns null
```

non-Primitive Data Type

Object

An object data type stores multiple values or data collection regarding properties and methods. It contains key-value pairs in its address. The properties of the object are written as key: value pairs. Each pair is separated by commas, enclosed in curly braces {}. The key must be a string, and the value can be of any data type.

For Example

```
var student = {firstName: 'john', class: 7};
```

Array

An array is a collection of values of the same data type. The elements in the array are indexed. The value of the index starts from 0. In JavaScript arrays are immutable.

Example:

```
var arr = [40,30,12];
```

Expressions and operators

JavaScript's expressions and operators, including assignment, comparison, arithmetic, bitwise, logical, string, ternary and more.

At a high level, an *expression* is a valid unit of code that resolves to a value. There are two types of expressions: those that have side effects (such as assigning values) and those that purely *evaluate*.

The expression x = 7 is an example of the first type. This expression uses the = *operator* to assign the value seven to the variable x. The expression itself evaluates to 7.

The expression 3 + 4 is an example of the second type. This expression uses the + operator to add 3 and 4 together and produces a value, 7. However, if it's not eventually part of a bigger construct (for example, a variable declaration like const z = 3 + 4), its result will be immediately discarded — this is usually a programmer mistake because the evaluation doesn't produce any effects.

JavaScript Operators

JavaScript includes operators same as other languages. An operator performs some operation on single or multiple operands (data value) and produces a result. For example, in 1 + 2, the + sign is an operator and 1 is left side operand and 2 is right side operand. The + operator performs the addition of two numeric values and returns a result.

JavaScript includes following categories of operators.

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Assignment Operators
- Conditional Operators
- Ternary Operator

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations between numeric operands.

Operator	Description
+	Adds two numeric operands.
-	Subtract right operand from left operand
*	Multiply two numeric operands.
/	Divide left operand by right operand.
%	Modulus operator. Returns remainder of two operands.
++	Increment operator. Increase operand value by one.
--	Decrement operator. Decrease value by one.

Example: Arithmetic Operation

```
let x = 5, y = 10;
let z = x + y; //performs addition and returns 15
z = y - x; //performs subtraction and returns 5
z = x * y; //performs multiplication and returns 50
z = y / x; //performs division and returns 2
z = x % 2; //returns division remainder 1
```

The ++ and -- operators are unary operators. It works with either left or right operand only. When used with the left operand, e.g., x++, it will increase the value of x when the program control goes to the next statement. In the same way, when it is used with the right operand, e.g., ++x, it will increase the value of x there only. Therefore, x++ is called post-increment, and ++x is called pre-increment.

Example: Post and Pre Increment/Decrement

```
let x = 5;
x++; //post-increment, x will be 5 here and 6 in the next line
++x; //pre-increment, x will be 7 here
x--; //post-decrement, x will be 7 here and 6 in the next line
--x; //pre-decrement, x will be 5 here
```

String Concatenation

The + operator performs concatenation operation when one of the operands is of string type. The following example demonstrates string concatenation even if one of the operands is a string.

Example: + Operator with String

```
let a = 5, b = "Hello ", c = "World!", d = 10;
a + b; //returns "5Hello "
b + c; //returns "Hello World!"
a + d; //returns 15
b + true; //returns "Hello true"
c - b; //returns NaN; - operator can only used with numbers
```

Comparison Operators

JavaScript provides comparison operators that compare two operands and return a boolean value true or false.

Operators	Description
==	Compares the equality of two operands without considering type.
===	Compares equality of two operands with type.
!=	Compares inequality of two operands.
>	Returns a boolean value true if the left-side value is greater than the right-side value; otherwise, returns false.
<	Returns a boolean value true if the left-side value is less than the right-side value; otherwise, returns false.
>=	Returns a boolean value true if the left-side value is greater than or equal to the right-side value; otherwise, returns false.
<=	Returns a boolean value true if the left-side value is less than or equal to the right-side value; otherwise, returns false.

Example: JavaScript Comparison Operators

```
let a = 5, b = 10, c = "5";
let x = a;
a == c; // returns true
a === c; // returns false
a == x; // returns true
a != b; // returns true
a > b; // returns false
a < b; // returns true
a >= b; // returns false
a <= b; // returns true
```

Logical Operators

In JavaScript, the logical operators are used to combine two or more conditions. JavaScript provides the following logical operators.

Operator	Description
&&	&& is known as AND operator. It checks whether two operands are non-zero or not (0, false, undefined, null or "" are considered as zero). It returns 1 if they are non-zero; otherwise, returns 0.
||	|| is known as OR operator. It checks whether any one of the two operands is non-zero or not (0, false, undefined, null or "" is considered as zero). It returns 1 if any one of of them is non-zero; otherwise, returns 0.
!	! is known as NOT operator. It reverses the boolean result of the operand (or condition). !false returns true, and !true returns false.

Example: Logical Operators

```
let a = 5, b = 10;
(a != b) && (a < b); // returns true
(a > b) || (a == b); // returns false
(a < b) || (a == b); // returns true
!(a < b); // returns false
!(a > b); // returns true
```

Assignment Operators

JavaScript provides the assignment operators to assign values to variables with less key strokes.

Assignment operators	Description
=	Assigns right operand value to the left operand.
+=	Sums up left and right operand values and assigns the result to the left operand.
-=	Subtract right operand value from the left operand value and assigns the result to the left operand.
*=	Multiply left and right operand values and assigns the result to the left operand.
/=	Divide left operand value by right operand value and assign the result to the left operand.
%=	Get the modulus of left operand divide by right operand and assign resulted modulus to the left operand.

Example: Assignment operators

```
let x = 5, y = 10, z = 15;
x = y; //x would be 10
x += 1; //x would be 6
x -= 1; //x would be 4
x *= 5; //x would be 25
x /= 5; //x would be 1
x %= 2; //x would be 1
```

Ternary Operator

JavaScript provides a special operator called ternary operator :? that assigns a value to a variable based on some condition. This is the short form of the if else condition.

Syntax:

```
<condition> ? <value1> : <value2>;
```

The ternary operator starts with conditional expression followed by the ? operator. The second part (after ? and before :) will be executed if the condition turns out to be true. Suppose, the condition returns false, then the third part (after :) will be executed.

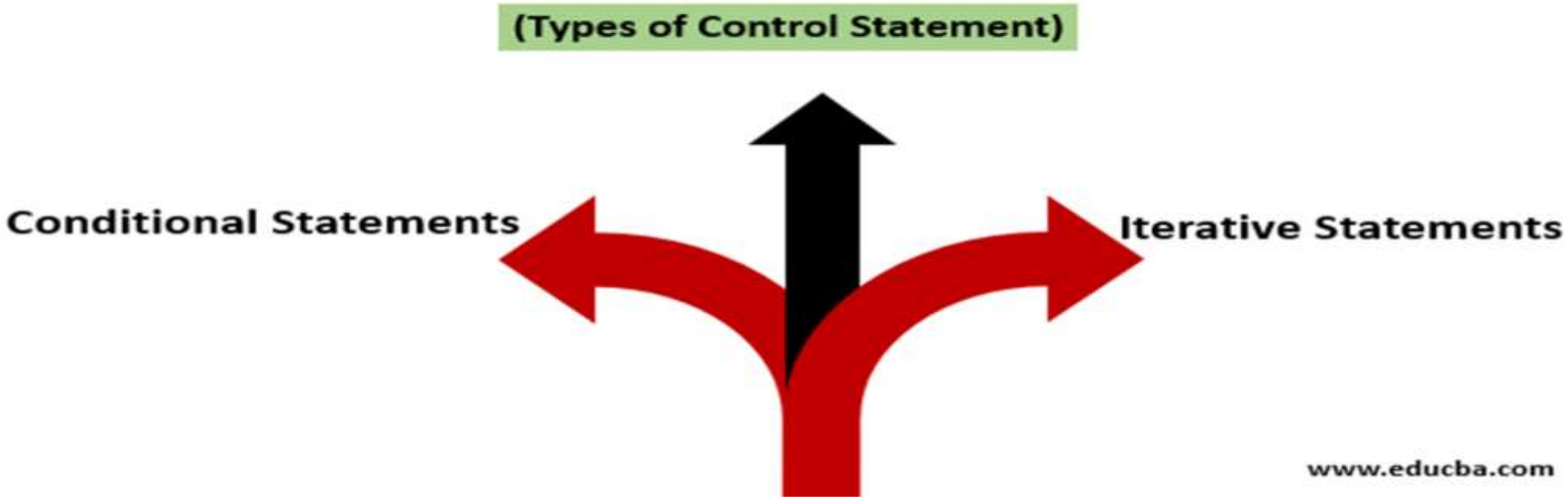
Example: Ternary operator

```
let a = 10, b = 5;
let c = a > b? a : b; // value of c would be 10
let d = a > b? b : a; // value of d would be 5
```

Introduction to Control Statement in JavaScript

Type of Control Statement in JavaScript

Every programming language, basically, has two types of control statements as follows:



Conditional Statements: based on an expression passed, a conditional statement makes a decision, which results in either YES or NO.

Iterative Statements (Loop): These statements continue to repeat until the expression or condition is satisfied.

1. Conditional Statements

Conditional statements in a program decide the next step based on the result. They result in either True or False. The program moves to the next step if a condition is passed and true. However, if the condition is false, the program moves to another step. These statements are executed only once, unlike loop statements.

Following are the different types of Conditional Statements:

IF

When you want to check for a specific condition with the IF condition, the inner code block executes if the provided condition is satisfied.

Syntax:

```
if (condition) {
//code block to be executed if condition is satisfied
}
```

Example : Simple Program for IF Statement

```
const num = 5;

if (num > 0) {
    console.log("The number is positive.");
};
```

IF-ELSE

The if-else statement will perform some action for a specific condition. If the condition meets then a particular code of action will be executed otherwise it will execute another code of action that satisfies that particular condition.

Syntax:

```
if (condition)
{
// code to be executed of condition is true
}
else {
// code to be executed of condition is false
}
```

As observed in an IF-ELSE statement, when the condition is satisfied, the first block of code executes, and if the condition isn’t satisfied, the second block of code executes.

```
let num = -10;
 if (num > 0)
    console.log("The number is positive.");
else
    console.log("The number is negative");
```

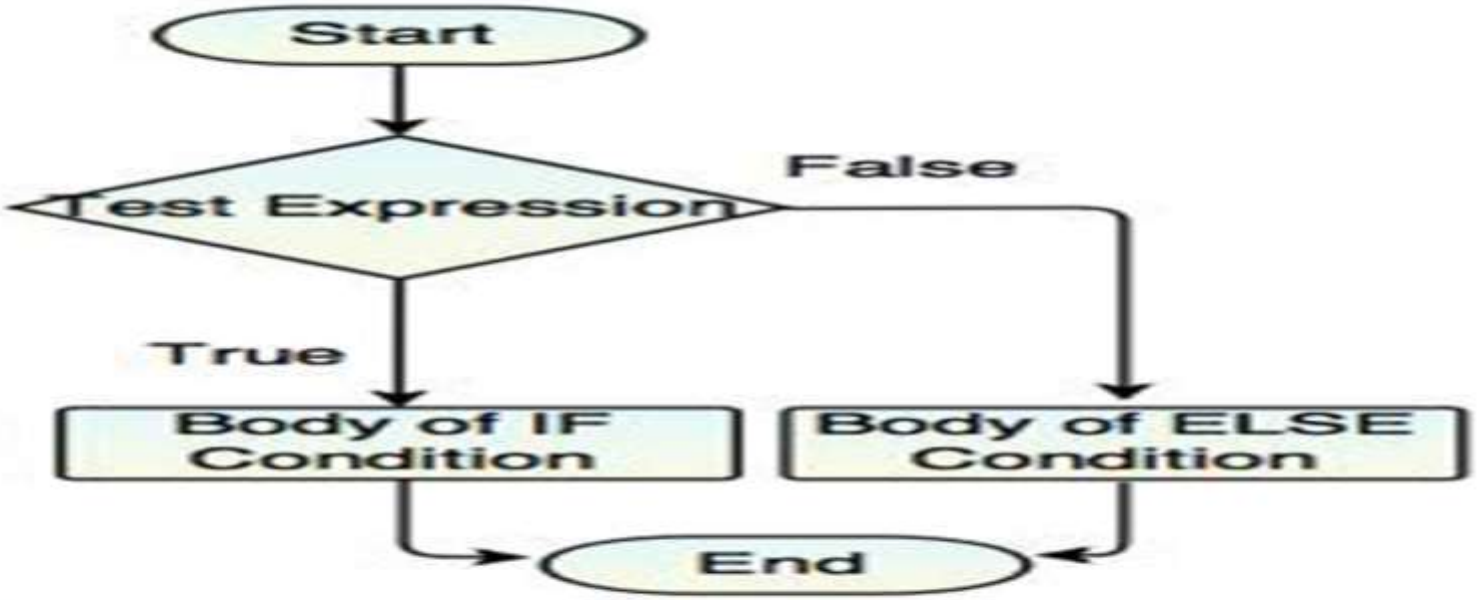


Fig. Flow Diagram of IF - ELSE Statement

If...else if...else Statement

You should use the if....else if...else statement if you want to select one of many sets of lines to execute.

Syntax

```
if (condition1)
 {
code to be executed if condition1 is true
 }
else if (condition2)
{
code to be executed if condition2 is true }
else
{ code to be executed if condition1 and condition2 are not true }
```

```
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time<10)
{
document.write("<b>Good morning</b>");
}
else if (time>10 && time<16)
{
document.write("<b>Good day</b>");
}
Else
{
document.write("<b>Hello World!</b>");
}
</script>
```

SWITCH

A switch statement is similar to IF and is useful when executing one code out of the multiple code block execution possibilities based on the result of the expression passed. Switch statements carry an expression, which is compared with values of the following cases, and once a match is found, the code associated with that case executes.

Syntax:

```
switch (expression) {
case a:
//code block to be executed
Break;
case b:
//code block to be executed
Break;
case n:
//code block to be executed
Break;
default:
//default code to be executed if none of the above case is executed
```

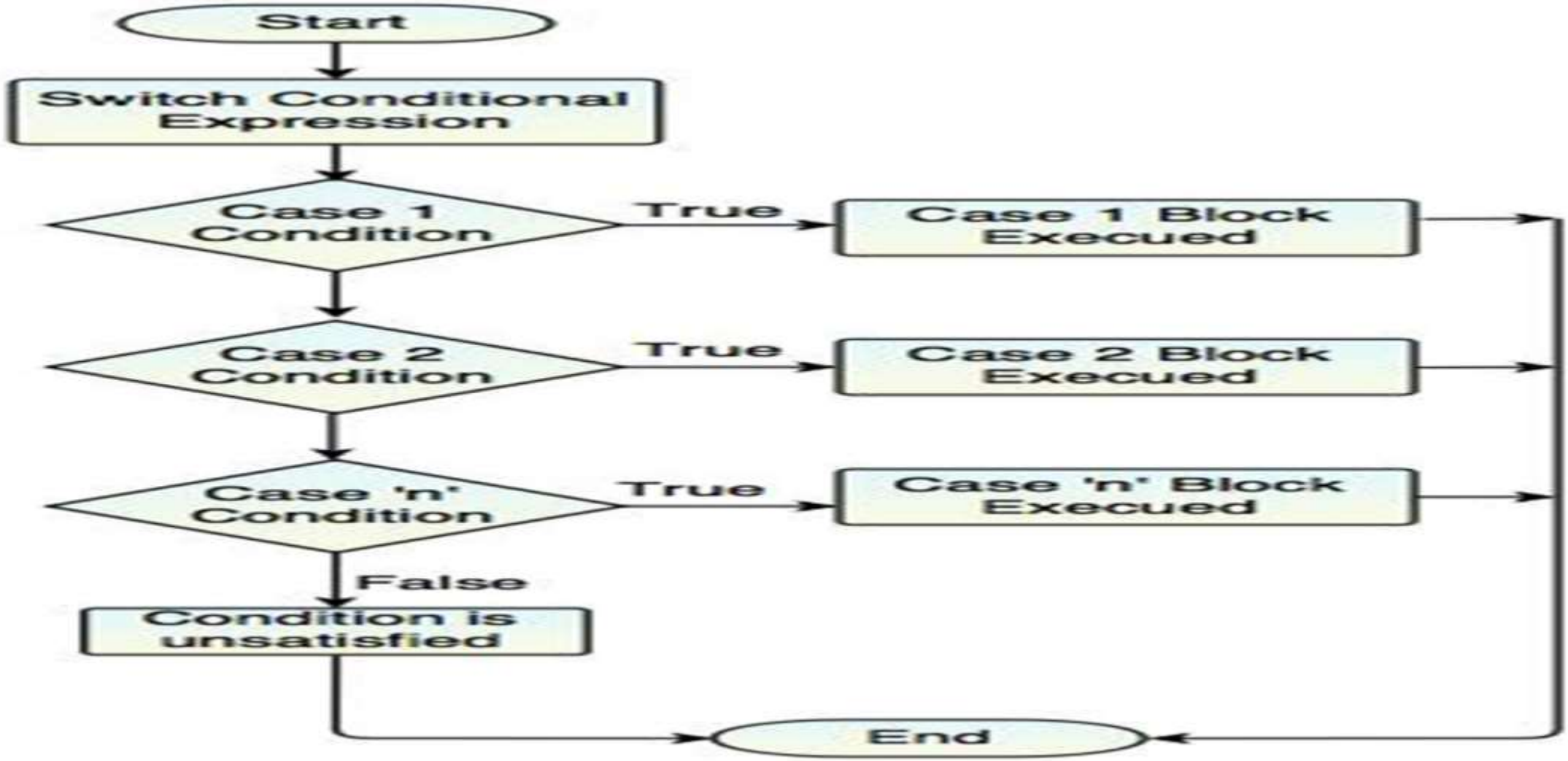


Fig. Flow Diagram of Switch Statement

```
let num = 5;
 switch (num) {
    case 0:
        console.log("Number is zero.");
        break;
    case 1:
        console.log("Nuber is one.");
        break;
    case 2:
        console.log("Number is two.");
        break;
    default:
        console.log("Number is greater than 2.");
};
```

The above code contains an expression at the beginning, checked and compared with the cases included. If the expression passed matches with case a, the code block inside the case is executed. The same applies to cases b and n, and when the expression passed matches with none of the cases mentioned, it code enters the default case, and the code under the default case is executed.

2. Iterative Statement

Looping is a powerful tool for any programming language to execute instructions repeatedly while the expression passed is satisfied. A basic example can be printing “Hello World” 10 times. Writing the same print statement with “Hello world“ for 10 straight times will be time-consuming and impact the execution time. And this is where looping comes in handy. There are three Iterative statements: WHILE, DO-WHILE, and FOR. Let’s understand each with syntax.

WHILE

A while loop evaluates the condition inside the parenthesis ().
If the condition evaluates to true, the code inside the while loop is executed.
The condition is evaluated again.
This process continues until the condition is false.
When the condition evaluates to false, the loop stops.

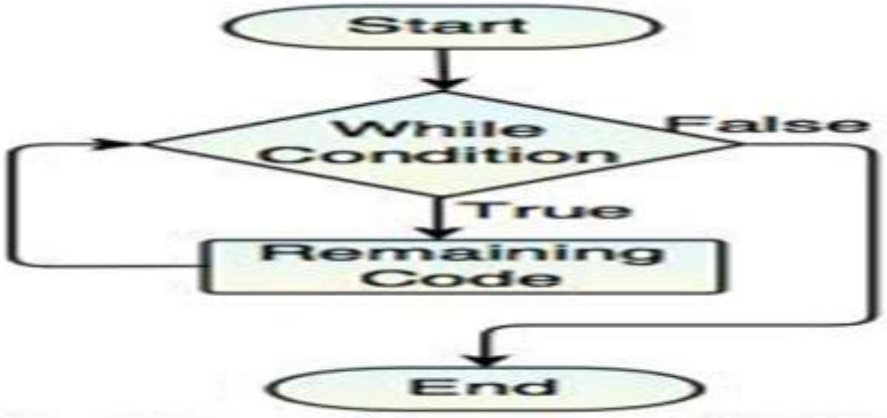


Fig. Flow Diagram of While Loop

Syntax:

```
while (condition)
{
//code block to be executed when condition is satisfied
}
```

Example:

```
let n = 0;
let x = 0;
while (n < 3) {
  n++;
  x += n;
}
```

Each iteration, the loop increments n and adds it to x. Therefore, x and n take on the following values:

After the first pass: n = 1 and x = 1

After the second pass: n = 2 and x = 3

After the third pass: n = 3 and x = 6

After completing the third pass, the condition n < 3 is no longer true, so the loop terminates

DO-WHILE

Syntax:

while

{

//code block to be executed when condition is satisfied

} (condition)

If the condition at the end is satisfied, the loop will repeat.

Here,

The body of the loop is executed at first. Then the **condition** is evaluated.

If the **condition** evaluates to true, the body of the loop inside the do statement is executed again.

The **condition** is evaluated once again.

If the **condition** evaluates to true, the body of the loop inside the do statement is executed again.

This process continues until the **condition** evaluates to false. Then the loop stops

```
// program to display numbers
let i = 1;
 const n = 5;
// do...while loop from 1 to 5
```

```
Do                                  Output
 {
console.log(i);                     1
 i++;                               2
}                                   3
 while(i <= n);                     4
                                    5
```

Here is how this program works.

Iteration	Variable	Condition: i <= n	Action
	i = 1		
	n = 5	not checked	1 is printed. i is increased to 2.
1st	i = 2		
	n = 5	true	2 is printed. i is increased to 3.
2nd	i = 3		
	n = 5	true	3 is printed. i is increased to 4.
3rd	i = 4		
	n = 5	true	4 is printed. i is increased to 5.
4th	i = 5		

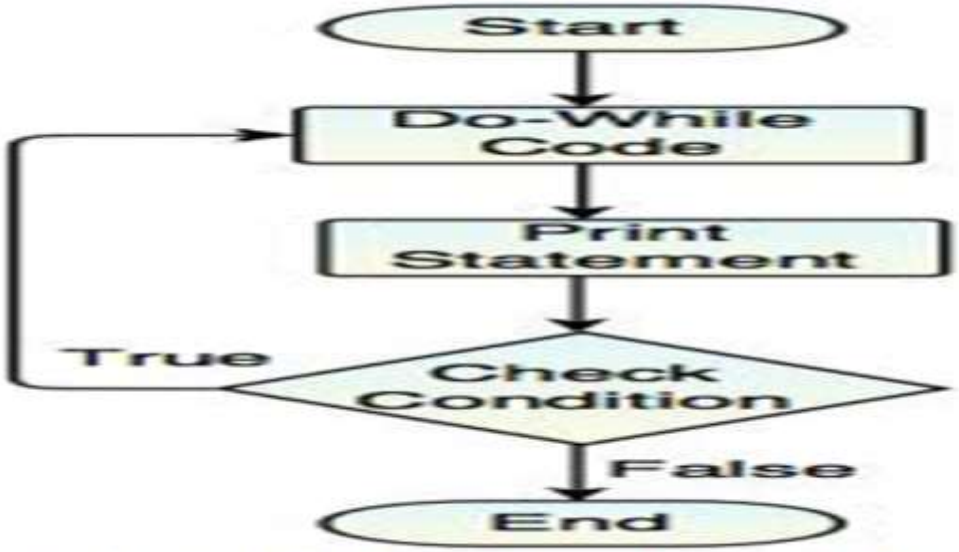


Fig. Flow Diagram of Do-While Loop

FOR

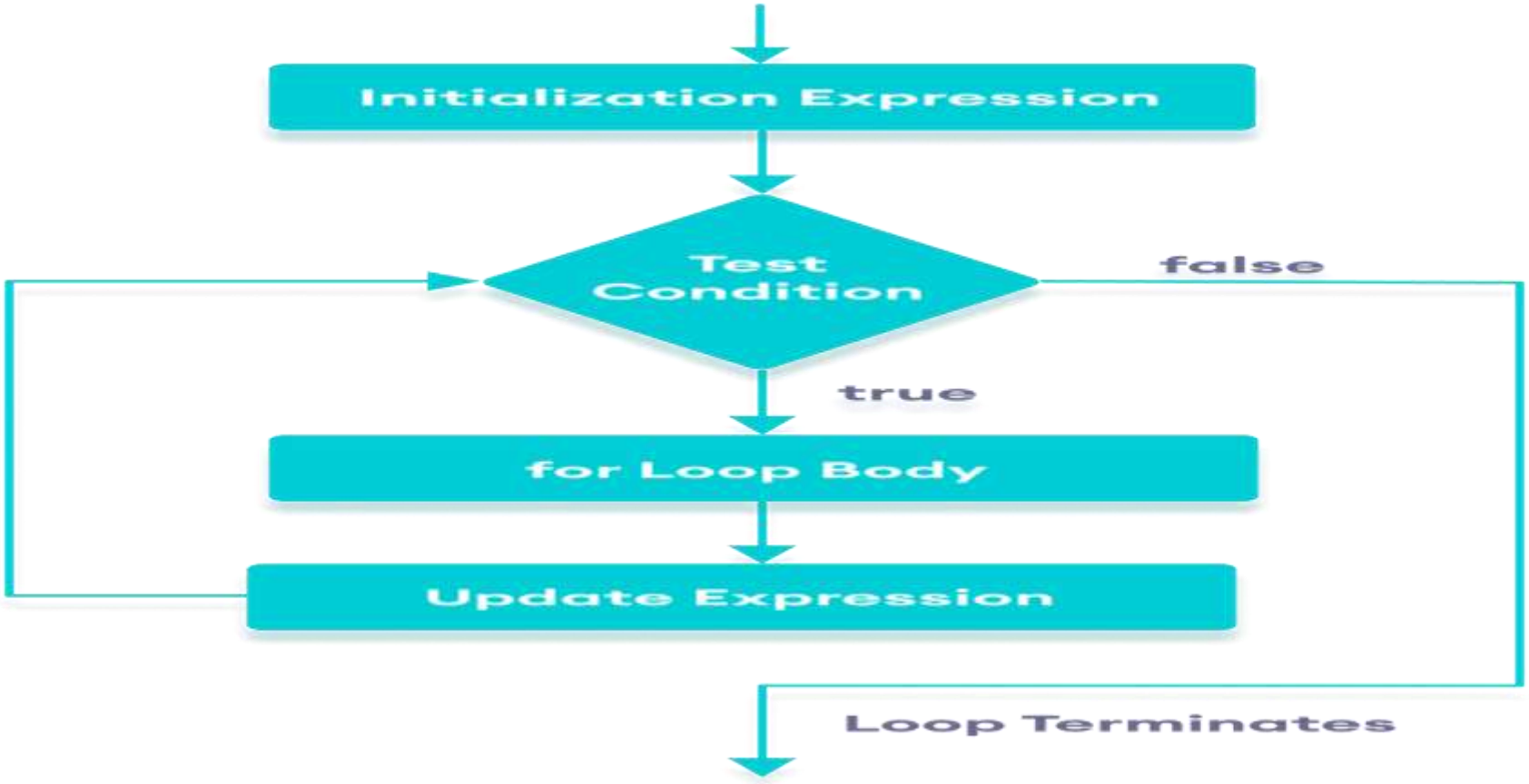
a for loop will execute a code block a number of times. Compared to other loops, FOR is shorter and easy to debug as it contains initialization, condition, and increment or decrement in a single line.

Syntax:

```
for (initialize; condition; increment/decrement)
{
//code block to be executed
}
```

.

With initialization, the loop starts by using a declared variable. Then, the condition part checks the exit condition for the loop. When this condition returns true, the code block inside is executed. If the condition returns false or fails, the control moves to the increment/decrement part, where the variable is updated. The values are updated until the condition is satisfied



Example 1: Display a Text Five Times

```
// program to display text 5 times
const n = 5;

// looping from i = 1 to 5
for (let i = 1; i <= n; i++) {
    console.log(`I love JavaScript.`);
```

Output

```
I love JavaScript.
I love JavaScript.
I love JavaScript.
I love JavaScript.
I love JavaScript
```

Here is how this program works.

Iteration	Variable	Condition: i <= n	Action
1st	i = 1		
	n = 5	true	I love JavaScript. is printed. i is increased to 2.
2nd	i = 2		
	n = 5	true	I love JavaScript. is printed. i is increased to 3.
3rd	i = 3		
	n = 5	true	I love JavaScript. is printed. i is increased to 4.
4th	i = 4		
	n = 5	true	I love JavaScript. is printed. i is increased to 5.
5th	i = 5		
	n = 5	true	I love JavaScript. is printed. i is increased to 6.
6th	i = 6		
	n = 5	false	The loop is terminated.