1. INTRODUCTION To Java Script

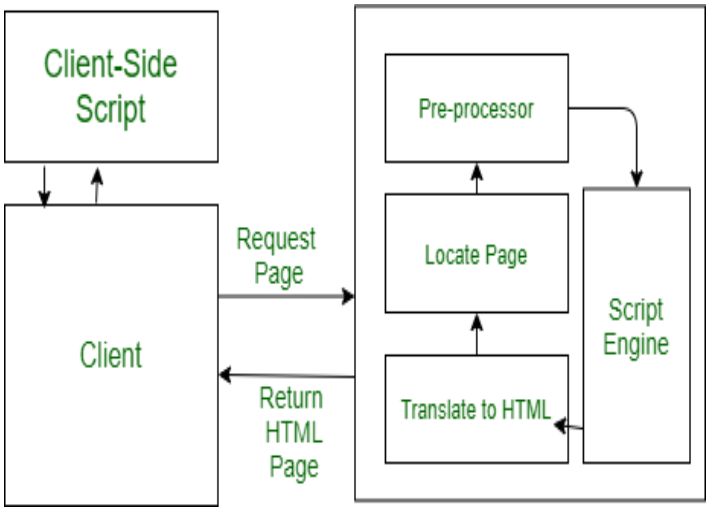
1. Web Scripting Fundamentals Server- Versus Client-Side Programming

The process of creating and embedding scripts in a web page is known as **web-scripting**. A script or a computer-script is a list of commands that are embedded in a web-page normally and are interpreted and executed by a certain program or scripting engine.

- Scripts may be written for a variety of purposes such as for automating processes on a local-computer or to generate web pages.
- The programming languages in which scripts are written are called scripting language, there are many scripting languages available today.
- Common scripting languages are VBScript, JavaScript, ASP, PHP, PERL, JSP etc.

Types of Script :

Scripts are broadly of following two type :



Web Scripting

Client-Side Scripts :

1. Client-side scripting is responsible for interaction within a web page. The client-side scripts are firstly downloaded at the client-end and then interpreted and executed by the browser (default browser of the system).
2. The client-side scripting is browser-dependent. i.e., the client-side browser must be scripting enables in order to run scripts
3. Client-side scripting is used when the client-side interaction is used. Some example uses of client-side scripting may be :
    - To get the data from user's screen or browser.
    - For playing online games.
    - Customizing the display of page in browser without reloading or reopening the page.
4. Here are some popular client-side scripting languages VBScript, JavaScript, Hypertext Processor(PHP).

Server-Side Scripts :

1. Server-side scripting is responsible for the completion or carrying out a task at the server-end and then sending the result to the client-end.
2. In server-side script, it doesn't matter which browser is being used at client-end, because the server does all the work.
3. Server-side scripting is mainly used when the information is sent to a server and to be processed at the server-end. Some sample uses of server-scripting can be :
    - Password Protection.

    - Browser Customization (sending information as per the requirements of client-end browser)
    - Form Processing
    - Building/Creating and displaying pages created from a database.
    - Dynamically editing, changing or adding content to a web-page.
4. Here are some popular server-side scripting languages PHP, Perl, ASP (Active Server Pages), JSP (Java Server Pages).

2. History, Features, JavaScript statements, A Simple Example, Code Editing Tools, The HTML Document, Keywords, Literals, JavaScript Values, Comments

A Brief JavaScript History

JavaScript was created at Netscape Communications by Brendan Eich in 1995.

Netscape and Eich designed JavaScript as a scripting language for use with the company's flagship web browser, Netscape Navigator. Initially known as LiveScript, Netscape changed the name to JavaScript so they could position it as a companion for the Java language, a product of their partner, Sun Microsystems.

JavaScript is in no way related to the Java programming language.

In 2008, the creation of Google's open-source Chrome V8, a high-performance JavaScript engine, provided a crucial turning point for JavaScript.The JavaScript standard, proposed for the first time as ECMAScript 1 in 1997, **Ecma International** (/ˈɛkmə/) is a nonprofit standards organization for information and communication systems.[1] It acquired its current name in 1994,

when the **European Computer Manufacturers Association** (**ECMA**) changed its name to reflect the organization's global reach and activities.

Current Version 12th Edition, ECMAScript

The subsequent proliferation of fast JavaScript engines made it possible for developers to build sophisticated browser-based applications with performance that competed with desktop and mobile applications.

Introduction

It began as a **Client-side Programming Language** run inside a web browser

JavaScript is the most widely used *client-side* programming language that lets you supercharge your HTML with interactivity, animation and dynamic visual effects for better User Interface and User Experience (UI/UX). It is:

- a small, lightweight, object-oriented, cross-platform, special-purpose scripting language meant to be run under a host environment (typically a web browser).
- a *client-side scripting language* to enrich web user-interfaces and create dynamic web pages (e.g., form input validation, and immediate response to user's actions).
- the engine that supports AJAX (Asynchronous JavaScript and XML), which generates renewed interest in JavaScript.

JavaScript works together with HTML/CSS. HTML provides the *contents* (or *structure*); CSS specifies the *presentation*; and JavaScript programs the *behavior.* Together, they enrich the UI/UX of the web users.

JavaScript is Now Everywhere with Node.js

JavaScript was originally created as a client-side web programming language, running in a web browser, to supercharge HTML. It has grown beyond the client-side programming.

With the introduction of Node.js (an open-source, cross-platform JavaScript run-time environment), you can run your JavaScript standalone or inside the server (instead of a browser). This allows you to use one single language for both the server-side and client-side programming.

JavaScript Features

Now let us see the features of JavaScript in detail:

1. **Object-Centered Script Language**

Object Centered Language features built in the object as Java Script has a window object. Some Common Examples of Object Centered languages are Java Script and Visual Basic etc. The object-centered languages are mostly used for features like Polymorphism which is a quality of taking an object in many forms. Use of Polymorphism within object-oriented programming requires whenever we use to represent reference of the parent class to an object of a child class.

2. **Client Edge Technology**

The client is basically a term used for Web Browser in respective of User. The data on the server gets uploaded by a client which later used by a user in the rendered form. The user gets access to the client through a web browser for surfing and interacting through websites. The client edge technology in Java Script allows the client to have full control over the content which is being updated in servers.

3. **Validation of User's Input**

Validation of User's Input is most commonly known as form validation, it allows users to interact with client through filling forms through web pages. The details in the form need to be correctly filled where form validation helps the client to validate the details entered by the user.

4. **Interpreter Centered**

Java Script is built with Interpreter Centered which allows the user to get the output without the use of Compiler. That means the input performed by the user gets rendered directly without the compiling of codes.

5. **Ability to Perform In Build Function**

Java Script has many In-Built Functions like isNAN (), Number (), parseFloat () and parseInt () etc. isNAN () Function is used to identify that input object is correct number format. parseFloat () function is used in the conversion of the object into a number. parseInt () Function is used to analyze strings.

6. **Case Sensitive Format**

The codes written in Java Script are Case Sensitive which explains that there will be no difference in the output whether the codes are written in Upper Case or Lower Case Format.

7.. **Light Weight and delicate**

Java Script Features Light Weight and delicate and codes written in JavaScript don't include variables and uses only objects to perform the operations.

8. **Statements Looping**

The statement looping is used to perform the same operations repeatedly. In this operation the same set of code run in repeat manner for a specific or unspecific set of time.

9.. **Handling Events**

The Java Script has the ability to control operations updated on servers. This is basically controlling the response on the website when the user tries to perform any operation the server handled by the client like clicking on links and options, interaction response over the website, etc.

3. JavaScript statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

JavaScript Programs

A computer program is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called statements.

A JavaScript program is a list of programming statements.

In HTML, JavaScript programs are executed by the web browser.

Program Example 1

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Statements</h2>

<p>In HTML, JavaScript statements are executed by the browser.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello Dolly.";
</script>

</body>
</html>
```

Output

JavaScript Statements

In HTML, JavaScript statements are executed by the browser.

Hello Dolly.

JavaScript programs (and JavaScript statements) are often called JavaScript code.

Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

Examples

```
let a, b, c;  // Declare 3 variables

a = 5;        // Assign the value 5 to a

b = 6;        // Assign the value 6 to b

c = a + b;    // Assign the sum of a and b to c
```

When separated by semicolons, multiple statements on one line are allowed:

```
a = 5; b = 6; c = a + b;
```

Hello World using Javascript

```
<html>
   <body>
      <script language = "javascript" type = "text/javascript">
         <!--
            document.write("Hello World!")
         //-->
      </script>
   </body>
</html>
```

There are many useful Javascript frameworks and libraries available:

- Angular
- React
- jQuery
- Vue.js
- Ext.js
- Ember.js
- Meteor
- Mithril
- Node.js
- Polymer
- Aurelia
- Backbone.js

It is really impossible to give a complete list of all the available Javascript frameworks and libraries. The Javascript world is just too large and too much new is happening.

Applications of Javascript Programming

As mentioned before, Javascript is one of the most widely used programming languages (Front-end as well as Back-end). It has it's presence in almost every area of software development. I'm going to list few of them here:

- **Client side validation** - This is really important to verify any user input before submitting it to the server and Javascript plays an important role in validting those inputs at front-end itself.
- **Manipulating HTML Pages** - Javascript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using javascript and modify your HTML to change its look and feel based on different devices and requirements.

- **User Notifications** - You can use Javascript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
- Back-end Data Loading - Javascript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.
- **Presentations** - JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides RevealJS and BespokeJS libraries to build a web-based slide presentations.
- **Server Applications** - Node JS is built on Chrome's Javascript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.

This list goes on, there are various areas where millions of software developers are happily using Javascript to develop great websites and others softwares.

Code editing Tools

A code editor is the place where programmers spend most of their time.

There are two main types of code editors: IDEs and lightweight editors. Many people use one tool of each type.

IDE

The term IDE (Integrated Development Environment) refers to a powerful editor with many features that usually operates on a “whole project.” As the name suggests, it’s not just an editor, but a full-scale “development environment.”

An IDE loads the project (which can be many files), allows navigation between files, provides autocompletion based on the whole project (not just the open file), and integrates with a version management system (like git), a testing environment, and other “project-level” stuff.

If you haven’t selected an IDE yet, consider the following options:

- Visual Studio Code (cross-platform, free).
- WebStorm (cross-platform, paid).

For Windows, there's also "Visual Studio", not to be confused with "Visual Studio Code". "Visual Studio" is a paid and mighty Windows-only editor, well-suited for the .NET platform. It's also good at JavaScript. There's also a free version Visual Studio Community.

Many IDEs are paid, but have a trial period. Their cost is usually negligible compared to a qualified developer's salary, so just choose the best one for you.

Lightweight editors

"Lightweight editors" are not as powerful as IDEs, but they're fast, elegant and simple.

They are mainly used to open and edit a file instantly.

The main difference between a "lightweight editor" and an "IDE" is that an IDE works on a project-level, so it loads much more data on start, analyzes the project structure if needed and so on. A lightweight editor is much faster if we need only one file.

In practice, lightweight editors may have a lot of plugins including directory-level syntax analyzers and autocompleters, so there's no strict border between a lightweight editor and an IDE.

The following options deserve your attention:

- Atom (cross-platform, free).
- Visual Studio Code (cross-platform, free).
- Sublime Text (cross-platform, shareware).
- Notepad++ (Windows, free).
- Vim and Emacs are also cool if you know how to use them.

The HTML Document

An HTML document is a file containing hypertext markup language that is formatted using HTML code. Study the definition and structure of an HTML document, types of HTML, and HTML examples.

Definition of an HTML Document

An HTML document is a file containing hypertext markup language. HTML code is based on tags, or hidden keywords, which provide instructions for formatting the document. A tag starts with an angle bracket and the 'less than' sign: '<'. The tag ends with an angle bracket and the 'greater than' sign '>'. Tags tell the processing program, often the web browser, what to do with the text. For example, to make the word 'Hello'

bold, you would use the opening bold tag and then the closing bold tag , like this:

Hello

HTML is defined by the World Wide Web Consortium, an organization that regulates standards for the Internet. Each version of HTML has a set of definitions. Note that HTML is not a programming language. While we often refer to HTML markup as HTML code, programming languages require the processing of logical statements and math. HTML allows the developer to make text documents look engaging and pleasant. In most cases, programming on an HTML document is done with JavaScript.

An HTML document is a file containing hypertext markup language that is formatted using HTML code.

Definition of an HTML Document

An HTML document is a file containing hypertext markup language. HTML code is based on tags, or hidden keywords, which provide instructions for formatting the document. A tag starts with an angle bracket and the 'less than' sign: '<'. The tag ends with an angle bracket and the 'greater than' sign '>'. Tags tell the processing program, often the web browser, what to do with the text. For example, to make the word 'Hello' bold, you would use the opening bold tag and then the closing bold tag , like this:

Hello

HTML is defined by the World Wide Web Consortium, an organization that regulates standards for the Internet. Each version of HTML has a set of definitions. Note that HTML is not a programming language. While we often refer to HTML markup as HTML code, programming languages require the processing of logical statements and math. HTML allows the developer to make text documents look engaging and pleasant. In most cases, programming on an HTML document is done with JavaScript.

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Example Explained

- The `<!DOCTYPE html>` declaration defines that this document is an HTML5 document
- The `<html>` element is the root element of an HTML page
- The `<head>` element contains meta information about the HTML page
- The `<title>` element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The `<body>` element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The `<h1>` element defines a large heading

- The <p> element defines a paragraph

Keywords

Keywords are reserved words in JavaScript that you cannot use to indicate variable labels or function names. There are a total of 63 keywords that JavaScript provides to programmers. All of them are shown in the below-mentioned table.

You can't use a keyword as an identifier in your JavaScript programs, its reserved words, and used to perform an internal operation.

abstract	**arguments**	**boolean**	**break**
byte	**case**	**catch**	**char**
const	**continue**	**debugger**	**default**
delete	**do**	**double**	**else**
eval	**false**	**final**	**finally**
float	**for**	**function**	**goto**
if	**implements**	**in**	**instanceof**
int	**interface**	**let**	**long**
native	**new**	**null**	**package**
private	**protected**	**public**	**return**

short	**static**	**switch**	**synchronized**
this	**throw**	**throws**	**transient**
true	**try**	**typeof**	**var**
void	**volatile**	**while**	**with**
yield			

```
const a = 'hello';
```

Here, `const` is a keyword that denotes that `a` is a constant.

Literals

JavaScript Literals are the fixed value that cannot be changed, you do not need to specify any type of keyword to write literals. Literals are often used to initialize variables in programming, names of variables are string literals.

A JavaScript Literal can be a numeric, string, floating-point value, a boolean value or even an object. In simple words, any value is literal, if you write a string "Studytonight" is a literal, any number like 7007 is a literal, etc.

JavaScript supports various types of literals which are listed below:

- Numeric Literal
- Floating-Point Literal
- Boolean Literal
- String Literal
- Array Literal
- Regular Expression Literal
- Object Literal

JavaScript Numeric Literal

- It can be expressed in the decimal(base 10), hexadecimal(base 16) or octal(base 8) format.
- Decimal numeric literals consist of a sequence of digits (0-9) without a leading 0(zero).
- Hexadecimal numeric literals include digits(0-9), letters (a-f) or (A-F).
- Octal numeric literals include digits (0-7). A leading 0(zero) in a numeric literal indicates octal format.

JavaScript Numeric Literals Example

```
120 // decimal literal

021434 // octal literal


0x4567 // hexadecimal literal
```

JavaScript Floating-Point Literal

- It contains a decimal point(.)
- A fraction is a floating-point literal
- It may contain an Exponent.

JavaScript Floating-Point Literal Example

```
6.99689 // floating-point literal

-167.39894 // negative floating-point literal
```

JavaScript Boolean Literal

Boolean literal supports two values only either true or false.

JavaScript Boolean Literal Example

```
true // Boolean literal

false // Boolean literal
```

JavaScript String Literal

A string literal is a combination of zero or more characters enclosed within a single(') or double quotation marks (").

JavaScript String Literal Example

```
"Study" // String literal

'tonight' // String literal
```

String literals can have some special characters too which are tabled below.

JavaScript Values

The specification makes an important distinction between values:

- *Primitive values* are the elements of the types undefined, null, boolean, number, bigint, string, symbol.
- All other values are *objects*.

In contrast to Java (that inspired JavaScript here), primitive values are not second-class citizens. The difference between them and objects is more subtle. In a nutshell:

- Primitive values: are atomic building blocks of data in JavaScript.
    - They are *passed by value*: when primitive values are assigned to variables or passed to functions, their contents are copied.
    - They are *compared by value*: when comparing two primitive values, their contents are compared.
- Objects: are compound pieces of data.
    - They are *passed by identity* (my term): when objects are assigned to variables or passed to functions, their *identities* (think pointers) are copied.
    - They are *compared by identity* (my term): when comparing two objects, their identities are compared.

Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments

Single line comments start with `//`.

Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

This example uses a single-line comment before each code line:

Example

```
// Change heading:

document.getElementById("myH").innerHTML = "My First Page";



// Change paragraph:

document.getElementById("myP").innerHTML = "My first paragraph.";
```

This example uses a single line comment at the end of each line to explain the code:

Example

```
let x = 5;       // Declare x, give it the value of 5

let y = x + 2;   // Declare y, give it the value of x + 2
```

Multi-line Comments

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

Example

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

It is most common to use single line comments.

Block comments are often used for formal documentation.

Using Comments to Prevent Execution

Using comments to prevent execution of code is suitable for code testing.

Adding `//` in front of a code line changes the code lines from an executable line to a comment.

This example uses // to prevent execution of one of the code lines:

Example

```
//document.getElementById("myH").innerHTML = "My First Page";

document.getElementById("myP").innerHTML = "My first paragraph.";
```

This example uses a comment block to prevent execution of multiple lines:

Example

```
/*

document.getElementById("myH").innerHTML = "My First Page";

document.getElementById("myP").innerHTML = "My first paragraph.";

*/
```

4. Variables and data types

A JavaScript variable is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Correct JavaScript variables

```
var x = 10;

var _value="sonoo";
```

Incorrect JavaScript variables

```
var  123=30;

var *aa=320;
```

Example of JavaScript variable

Let's see a simple example of JavaScript variable.

```
<script>

var x = 10;

var y = 20;

var z=x+y;

document.write(z);

</script>
```

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

```
<script>

function abc(){

var x=10;//local variable

}

</script>
```

Or,

```
<script>

if(10<13){

var y=20;//JavaScript local variable

}

</script>
```

JavaScript global variable

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

```
<script>
var data=200;//global variable
function a(){
document.writeln(data);
}
function b(){
document.writeln(data);
}
a();//calling JavaScript function
b();
</script>
```

Javascript Data Types:

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

```
1. var a=40;//holding number
2. var b="Rahul";//holding string
```

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

Example

```
<!DOCTYPE html>

<html>

<body>



<h2>JavaScript Data Types</h2>



<p>JavaScript has dynamic types. This means that the same variable can be used to hold different data types:</p>



<p id="demo"></p>



<script>

let x;         // Now x is undefined

x = 5;         // Now x is a Number

x = "John";    // Now x is a String



document.getElementById("demo").innerHTML = x;
```

```
</script>


</body>

</html>
```

Output

JavaScript Data Types

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

John

JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Strings</h2>

<p>Strings are written with quotes. You can use single or double quotes:</p>

<p id="demo"></p>

<script>
let carName1 = "Volvo XC60";
```

```
let carName2 = 'Volvo XC60';

document.getElementById("demo").innerHTML =
carName1 + "<br>" +
carName2;
</script>

</body>
</html>
```

Output

JavaScript Strings

Strings are written with quotes. You can use single or double quotes:

Volvo XC60

Volvo XC60

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

```
let answer1 = "It's alright";             // Single quote inside double quotes

let answer2 = "He is called 'Johnny'";    // Single quotes inside double quotes

let answer3 = 'He is called "Johnny"';    // Double quotes inside single quotes
```

Example

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>JavaScript Strings</h2>

<p>You can use quotes inside a string, as long as they don't match the quotes surrounding the string:</p>

<p id="demo"></p>

<script>
let answer1 = "It's alright";
let answer2 = "He is called 'Johnny'";
let answer3 = 'He is called "Johnny"';

document.getElementById("demo").innerHTML =
answer1 + "<br>" +
answer2 + "<br>" +
answer3;
</script>

</body>
</html>
```

Output:

JavaScript Strings

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

It's alright

He is called 'Johnny'

He is called "Johnny"

JavaScript Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

Example

```
let x1 = 34.00;      // Written with decimals

let x2 = 34;         // Written without decimals
```

Extra large or extra small numbers can be written with scientific (exponential) notation:

Example

```
let y = 123e5;       // 12300000

let z = 123e-6;      // 0.000123
```

Example 1:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Numbers can be written with, or without decimals:</p>
```

```
<p id="demo"></p>

<script>
let x1 = 34.00;
let x2 = 34;
let x3 = 3.14;

document.getElementById("demo").innerHTML =
x1 + "<br>" + x2 + "<br>" + x3;
</script>

</body>
</html>
```

Output:

JavaScript Numbers

Numbers can be written with, or without decimals:

34

34

3.14

Example 2:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Numbers</h2>

<p>Extra large or extra small numbers can be written with scientific (exponential) notation:</p>

<p id="demo"></p>

<script>
let y = 123e5;
```

```
let z = 123e-5;

document.getElementById("demo").innerHTML =
y + "<br>" + z;
</script>

</body>
</html>
```

Output:

JavaScript Numbers

Extra large or extra small numbers can be written with scientific (exponential) notation:

12300000

0.00123

JavaScript Booleans

Booleans can only have two values: `true` or `false`.

Example

```
let x = 5;

let y = 5;

let z = 6;

(x == y)       // Returns true

(x == z)       // Returns false
```

Booleans are often used in conditional testing.

Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Booleans</h2>

<p>Booleans can have two values: true or false:</p>

<p id="demo"></p>

<script>
let x = 5;
let y = 5;
let z = 6;

document.getElementById("demo").innerHTML =
(x == y) + "<br>" + (x == z);
</script>

</body>
</html>
```

Output:

JavaScript Booleans

Booleans can have two values: true or false:

true

false

JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called `cars`, containing three items (car names):

Example

```
const cars = ["Saab", "Volvo", "BMW"];
```

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Array indexes are zero-based, which means the first item is [0].</p>

<p id="demo"></p>

<script>
const cars = ["Saab","Volvo","BMW"];

document.getElementById("demo").innerHTML = cars[0];
</script>

</body>
</html>
```

Output:

JavaScript Arrays

Array indexes are zero-based, which means the first item is [0].

Saab

JavaScript Objects

JavaScript objects are written with curly braces `{}`.

Object properties are written as name:value pairs, separated by commas.

Example

```
const person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p id="demo"></p>

<script>
const person = {
  firstName : "John",
  lastName  : "Doe",
  age     : 50,
  eyeColor  : "blue"
};

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

Output:

JavaScript Objects

John is 50 years old.

The typeof Operator

You can use the JavaScript `typeof` operator to find the type of a JavaScript variable.

The `typeof` operator returns the type of a variable or an expression:

Example

```
typeof ""              // Returns "string"

typeof "John"          // Returns "string"

typeof "John Doe"      // Returns "string"
```

Example

```
typeof 0               // Returns "number"

typeof 314             // Returns "number"

typeof 3.14            // Returns "number"

typeof (3)             // Returns "number"

typeof (3 + 4)         // Returns "number"
```

Undefined

In JavaScript, a variable without a value, has the value `undefined`. The type is also `undefined`.

Example

```
let car;

 // Value is undefined, type is undefined

var car;
```

Any variable can be emptied, by setting the value to `undefined`. The type will also be `undefined`.

Example

```
car = undefined;    // Value is undefined, type is undefined
```

Empty Values

An empty value has nothing to do with `undefined`.

An empty string has both a legal value and a type.

Example

```
let car = "";    // The value is "", the typeof is "string"
```

5. Expressions and Operators

Expressions

An expression is any valid set of literals, variables, operators, and expressions that evaluates to a single value. The value may be a number, a string, or a logical value. Conceptually, there are two types of expressions: those that assign a value to a variable, and those that simply have a value. For example, the expression

x = 7

is an expression that assigns x the value 7. This expression itself evaluates to 7. Such expressions use *assignment operators.* On the other hand, the expression

3 + 4

simply evaluates to 7; it does not perform an assignment. The operators used in such expressions are referred to simply as *operators*.

JavaScript has the following kinds of expressions:

- **Arithmetic: evaluates to a number, for example**
- **String: evaluates to a character string, for example "Fred" or "234"**
- **Logical: evaluates to true or false**

The special keyword null denotes a null value. In contrast, variables that have not been assigned a value are *undefined*, and cannot be used without a run-time error.

Conditional Expressions

A conditional expression can have one of two values based on a condition. The syntax is

(*condition*) ? *val1* : *val2*

If *condition* is true, the expression has the value of *val1*, Otherwise it has the value of *val2*. You can use a conditional expression anywhere you would use a standard expression.

For example,

status = (age >= 18) ? "adult" : "minor"

This statement assigns the value "adult" to the variable status if age is eighteen or greater. Otherwise, it assigns the value "minor" to status.

Let us take a simple expression 4 + 5 is equal to 9. Here 4 and 5 are called operands and '+' is called the operator. JavaScript supports the following types of operators.

- **Arithmetic Operators**
- **Comparison Operators**
- **Logical (or Relational) Operators**
- **Assignment Operators**
- **Conditional (or ternary) Operators**

Lets have a look on all operators one by one.

Arithmetic Operators

JavaScript supports the following arithmetic operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	+ (Addition) Adds two operands Ex: A + B will give 30
2	- (Subtraction) Subtracts the second operand from the first

	Ex: A - B will give -10
3	* (Multiplication) Multiply both operands Ex: A * B will give 200
4	/ (Division) Divide the numerator by the denominator Ex: B / A will give 2
5	% (Modulus) Outputs the remainder of an integer division Ex: B % A will give 0
6	++ (Increment) Increases an integer value by one Ex: A++ will give 11

7	-- (Decrement) Decreases an integer value by one Ex: A-- will give 9

Note − Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Example

```
<html>
   <body>

      <script>
            var a = 33;
            var b = 10;
            var c = "Test";
            var linebreak = "<br />";

            document.write("a + b = ");
            result = a + b;
            document.write(result);
            document.write(linebreak);

            document.write("a - b = ");
            result = a - b;
            document.write(result);
            document.write(linebreak);

            document.write("a / b = ");
            result = a / b;
            document.write(result);
            document.write(linebreak);

            document.write("a % b = ");
            result = a % b;
            document.write(result);
```

```
            document.write(linebreak);

            document.write("a + b + c = ");
            result = a + b + c;
            document.write(result);
            document.write(linebreak);

            a = ++a;
            document.write("++a = ");
            result = ++a;
            document.write(result);
            document.write(linebreak);

            b = --b;
            document.write("--b = ");
            result = --b;
            document.write(result);
            document.write(linebreak);
               </script>

      Set the variables to different values and then try...
   </body>
</html>
```

Output

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
Set the variables to different values and then try...
```

Comparison Operators

JavaScript supports the following comparison operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	= = (Equal) Checks if the value of two operands are equal or not, if yes, then the condition becomes true. Ex: (A == B) is not true.
2	!= (Not Equal) Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. Ex: (A != B) is true.
3	> (Greater than) Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.

	Ex: (A > B) is not true.
4	< (Less than) Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. Ex: (A < B) is true.
5	>= (Greater than or Equal to) Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. Ex: (A >= B) is not true.
6	<= (Less than or Equal to) Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true. Ex: (A <= B) is true.

Example

The following code shows how to use comparison operators in JavaScript.

```
<html>
```

```
   <body>
      <script type = "text/javascript">
         <!--
            var a = 10;
            var b = 20;
            var linebreak = "<br />";

            document.write("(a == b) => ");
            result = (a == b);
            document.write(result);
            document.write(linebreak);

            document.write("(a < b) => ");
            result = (a < b);
            document.write(result);
            document.write(linebreak);

            document.write("(a > b) => ");
            result = (a > b);
            document.write(result);
            document.write(linebreak);

            document.write("(a != b) => ");
            result = (a != b);
            document.write(result);
            document.write(linebreak);

            document.write("(a >= b) => ");
            result = (a >= b);
            document.write(result);
            document.write(linebreak);

            document.write("(a <= b) => ");
            result = (a <= b);
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      Set the variables to different values and different
operators and then try...
   </body>
</html>
```

Output

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
a <= b) => true
Set the variables to different values and different operators and then try...
```

Logical (or Relational) Operators

JavaScript supports the following logical operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	&& (Logical AND) If both the operands are non-zero, then the condition becomes true. Ex: (A && B) is true.
2	|| (Logical OR) If any of the two operands are non-zero, then the condition becomes true. Ex: (A || B) is true.

3	! (Logical NOT) Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. Ex: ! (A && B) is false.

Assignment Operators

JavaScript supports the following assignment operators −

Sr.No.	Operator & Description
1	= (Simple Assignment) Assigns values from the right side operand to the left side operand Ex: C = A + B will assign the value of A + B into C
2	+= (Add and Assignment) It adds the right operand to the left operand and assigns the result to the left operand. Ex: C += A is equivalent to C = C + A

3	−= (Subtract and Assignment) It subtracts the right operand from the left operand and assigns the result to the left operand. Ex: C -= A is equivalent to C = C - A
4	*= (Multiply and Assignment) It multiplies the right operand with the left operand and assigns the result to the left operand. Ex: C *= A is equivalent to C = C * A
5	/= (Divide and Assignment) It divides the left operand with the right operand and assigns the result to the left operand. Ex: C /= A is equivalent to C = C / A
6	%= (Modules and Assignment) It takes modulus using two operands and assigns the result to the left operand.

	Ex: C %= A is equivalent to C = C % A

Example

Try the following code to implement assignment operator in JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 33;
            var b = 10;
            var linebreak = "<br />";

            document.write("Value of a => (a = b) => ");
            result = (a = b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a += b) => ");
            result = (a += b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a -= b) => ");
            result = (a -= b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a *= b) => ");
            result = (a *= b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a /= b) => ");
            result = (a /= b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a %= b) => ");
            result = (a %= b);
```

```
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

Output

```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0
Set the variables to different values and different operators
and then try...
```

Conditional (or ternary) Operators

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Sr.No.	Operator and Description

1	? : (Conditional) If Condition is true? Then value X : Otherwise value Y

Example

Try the following code to understand how the Conditional Operator works in JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 10;
            var b = 20;
            var linebreak = "<br />";

            document.write ("((a > b) ? 100 : 200) => ");
            result = (a > b) ? 100 : 200;
            document.write(result);
            document.write(linebreak);

            document.write ("((a < b) ? 100 : 200) => ");
            result = (a < b) ? 100 : 200;
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

Output

```
((a > b) ? 100 : 200) => 200
((a < b) ? 100 : 200) => 100
Set the variables to different values and different operators
and then try...
```

6. If else statement, if else if statement, nested if statement, switch case

Decision Making in programming is similar to decision making in real life. In programming also we face some situations where we want a certain block of code to be executed when some condition is fulfilled.
A programming language uses control statements to control the flow of execution of the program based on certain conditions. These are used to cause the flow of execution to advance and branch based on changes to the state of a program.

JavaScript's conditional statements:

- if
- if-else
- nested-if
- if-else-if

  These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

if: if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.
Syntax:

```
if(condition)
{
   // Statements to execute if
   // condition is true
}
```

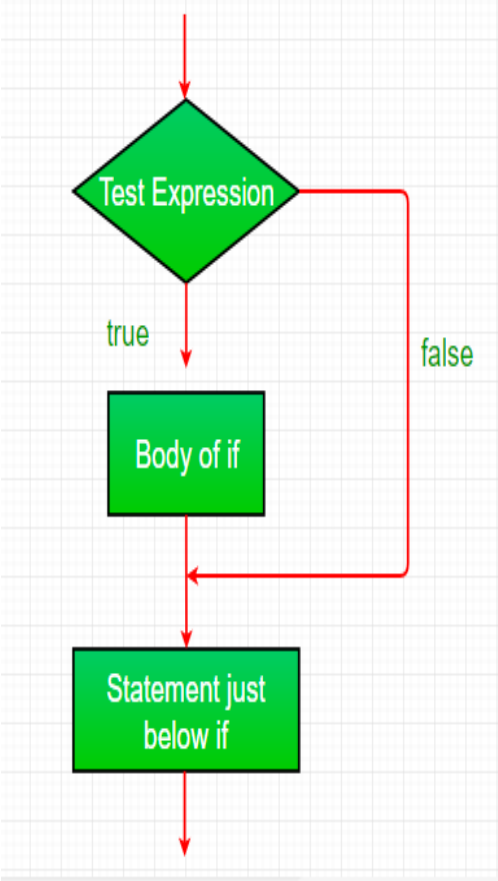
Here, **condition** after evaluation will be either true or false. if statement accepts boolean values – if the value is true then it will execute the block of statements under it. If we do not provide the curly braces '{' and '}' after **if(condition)** then by default if statement will consider the immediate one statement to be inside its block. For example,

```
if(condition)
   statement1;
```

```
    statement2;

// Here if the condition is true, if block
// will consider only statement1 to be inside
    // its block.
```

Flow chart:



Example:

```
<script type = "text/javaScript">

// JavaScript program to illustrate If
statement

var i = 10;

if (i > 15)
  document.write("10 is less than 15");

// This statement will be executed
// as if considers one statement by default
document.write("I am Not in if");

< /script>
```

Output:

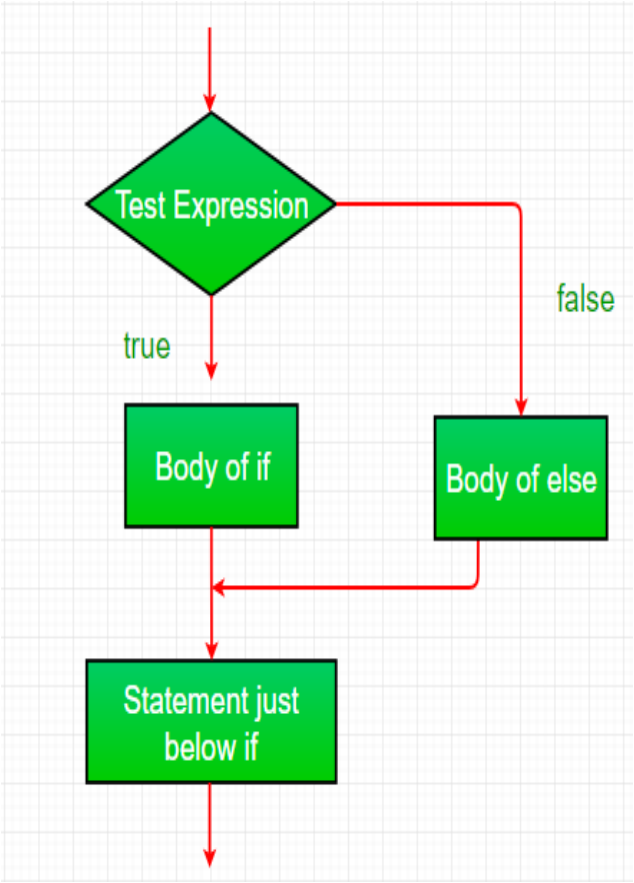
```
I am Not in if
```

-

if-else: The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

Syntax:

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

- 

Example:

```
<script type = "text/javaScript">

// JavaScript program to illustrate If-else
statement

var i = 10;

if (i < 15)
  document.write("10 is less than 15");
else
  document.write("I am Not in if");
```

```
< /script>
```

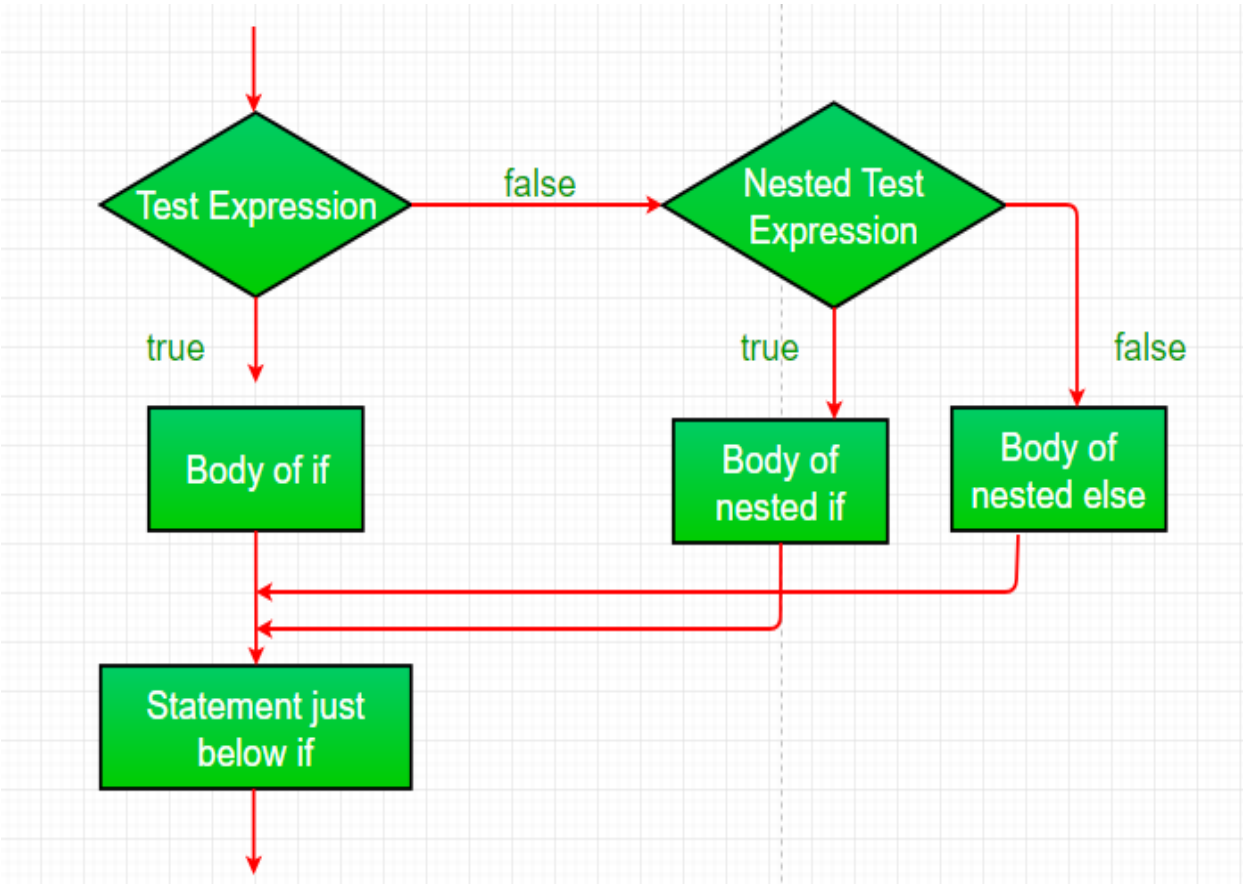
- Output:

```
i is smaller than 15
```

nested-if: A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement. Yes, JavaScript allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

Syntax:

```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
   {
      // Executes when condition2 is true
   }
}
```

- 
Test Expression
false
Nested Test Expression
true
true
false
Body of if
Body of nested if
Body of nested else
Statement just below if

Example:

```
<script type = "text/javaScript">

// JavaScript program to illustrate nested-if
statement

var i = 10;

if (i == 10) {

  // First if statement
  if (i < 15)
    document.write("i is smaller than 15");

  // Nested - if statement
  // Will only be executed if statement above
  // it is true
  if (i < 12)
    document.write("i is smaller than 12 too");
  else
    document.write("i is greater than 15");
}
< /script>
```

Output:

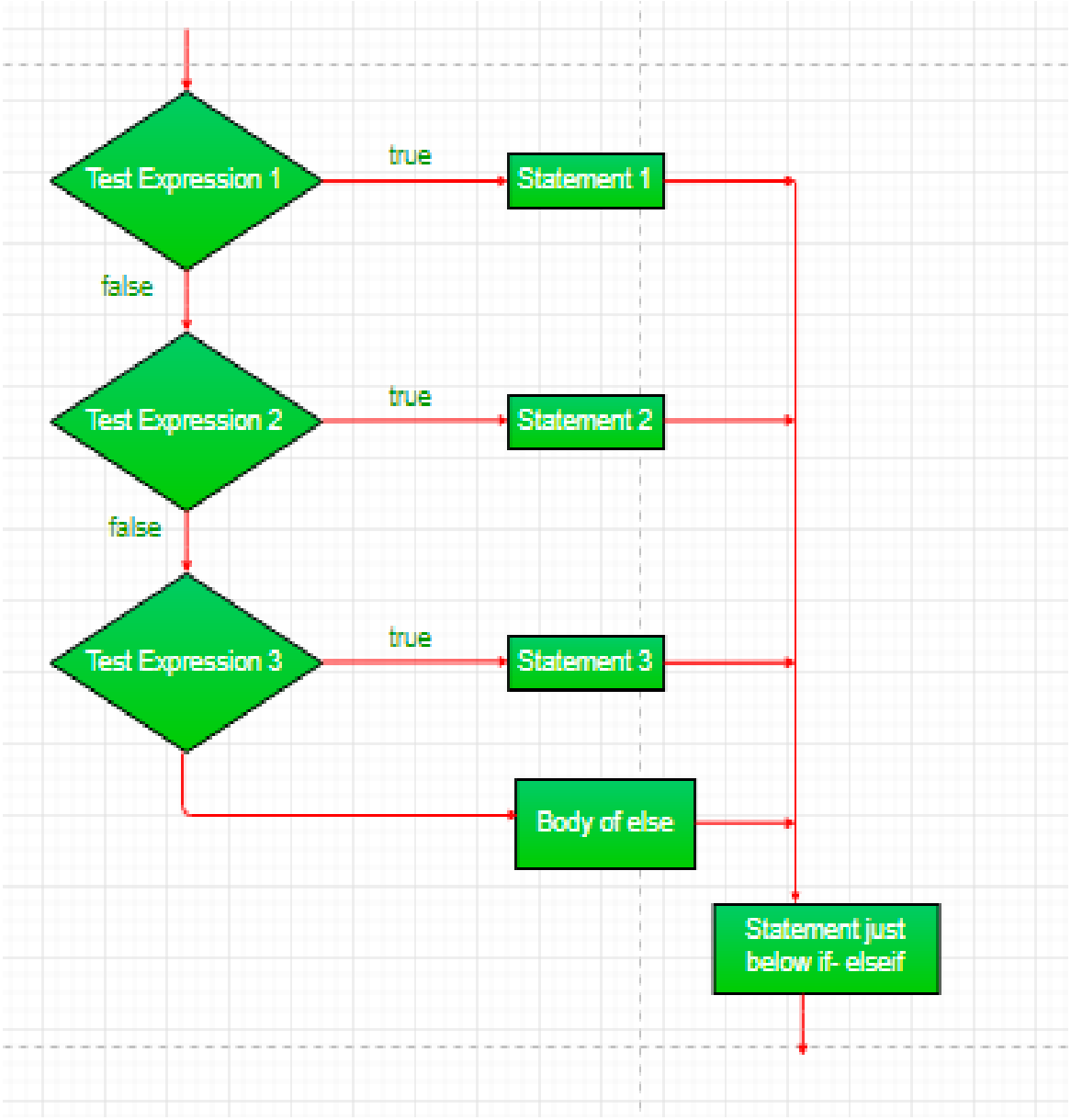
```
i is smaller than 15
i is smaller than 12 too
```

-

if-else-if ladder: Here, a user can decide among multiple options.The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
```

```
else
    statement;
```

- 

Example:

```
<script type = "text/javaScript">
// JavaScript program to illustrate nested-if
statement

var i = 20;

if (i == 10)
  document.write("i is 10");
else if (i == 15)
  document.write("i is 15");
else if (i == 20)
  document.write("i is 20");
else
  document.write("i is not present");
< /script>
```

Output:

```
i is 20
```

Switch Statement

Use the `switch` statement to select one of many code blocks to be executed.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

Example

The `getDay()` method returns the weekday as a number between 0 and 6.

(Sunday=0, Monday=1, Tuesday=2 ..)

This example uses the weekday number to calculate the weekday name:

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
```

The result of day will be:

```
Monday
```

Example:

```
<script type = "text/javascript">

    // JavaScript program to illustrate
switch-case
    let i = 9;

    switch (i)
    {
       case 0:
           console.log("i is zero.");
           break;
       case 1:
           console.log("i is one.");
           break;
       case 2:
           console.log("i is two.");
           break;
       default:
           console.log("i is greater than
2.");
    }

</script>
```

Output:

```
i is greater than 2.
```

7. **Loop statement – for loop, for ---in loop, while loop, do – while loop, continue statement**

The JavaScript loops are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop
4. for-in loop

---

1) JavaScript For loop

The JavaScript for loop *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

```
for (initialization; condition; increment)

{

   code to be executed

}
```

Let's see the simple example of for loop in javascript.

```
<script>

for (i=1; i<=5; i++)

{

document.write(i + "<br/>")
```

```
}
</script>
```

Output:

```
1
2
3
4
5
```

JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

```
while (condition)
{
   code to be executed
}
```

Let's see the simple example of while loop in javascript.

```
<script>
var i=11;
while (i<=15)
```

```
{

document.write(i + "<br/>");

i++;

}

</script>
```

Test it Now

Output:

```
11
12
13
14
15
```

3) JavaScript do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

```
do{

   code to be executed

}while (condition);
```

Let's see the simple example of do while loop in javascript.

```
<script>

var i=21;

do{
```

```
document.write(i + "<br/>");

i++;

}while (i<=25);

</script>
```

Output:

```
21
22
23
24
25
```

4) JavaScript for in loop

The **JavaScript for in loop** is used *to iterate the properties of an object*. We will discuss about it later.

The JavaScript `for in` statement loops through the properties of an Object:

Syntax

```
for (key in object) {

  // code block to be executed

}
```

Example

```
const person = {fname:"John", lname:"Doe", age:25};
```

```
let text = "";

for (let x in person) {

  text += person[x];

}
```

Example Explained

- The for in loop iterates over a person object
- Each iteration returns a key (x)
- The key is used to access the value of the key
- The value of the key is person[x]

For In Over Arrays

The JavaScript `for in` statement can also loop over the properties of an Array:

Syntax

```
for (variable in array) {

  code

}
```

Example

```
const numbers = [45, 4, 9, 16, 25];
```

```
let txt = "";

for (let x in numbers) {

  txt += numbers[x];

}
```

8. **Inserting the JavaScript into an HTML document**

   There are following three ways in which users can add JavaScript to HTML pages.

   using external java script files with examples

External JavaScript file

We can create external JavaScript file and embed it in many html page.

It provides **code re usability** because single JavaScript file can be used in several html pages.

An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

Let's create an external JavaScript

file that prints Hello Javatpoint in a alert dialog box.

message.js

```
1. function msg(){
2.  alert("Hello Javatpoint");
3. }
```

Let's include the JavaScript file into html

page. It calls the JavaScript function
on button click.

index.html

```
1. <html>
2. <head>
```

```
3. <script type="text/javascript" src="message.js"></script>
4. </head>
5. <body>
6. <p>Welcome to JavaScript</p>
7. <form>
8. <input type="button" value="click" onclick="msg()"/>
9. </form>
10. </body>
11. </html>
```

Advantages of External JavaScript

There will be following benefits if a user creates an external javascript:

1. It helps in the reusability of code in more than one HTML file.
2. It allows easy code readability.
3. It is time-efficient as web browsers cache the external js files, which further reduces the page loading time.
4. It enables both web designers and coders to work with html and js files parallelly and separately, i.e., without facing any code conflictions.
5. The length of the code reduces as only we need to specify the location of the js file.

Disadvantages of External JavaScript

There are the following disadvantages of external files:

1. The stealer may download the coder's code using the url of the js file.
2. If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.

3. The web browser needs to make an additional http request to get the js code.
4. A tiny to a large change in the js code may cause unexpected results in all its dependent files.
5. We need to check each file that depends on the commonly created external javascript file.
6. If it is a few lines of code, then better to implement the internal javascript code.