CREDO SYSTEMZ
simplifying IT

ANGULAR

JavaScript - Training

Agenda

- Introduction to JavaScript
- Why JavaScript?
- How to Add JavaScript?
- Difference between – HTML/CSS/JS
- Understanding DOM
- DOM Methods
- Events & Functions
- Variables & DataTypes
- Operators
- Conditional Statements
- Loops
- Arrarys

What is JavaScript?

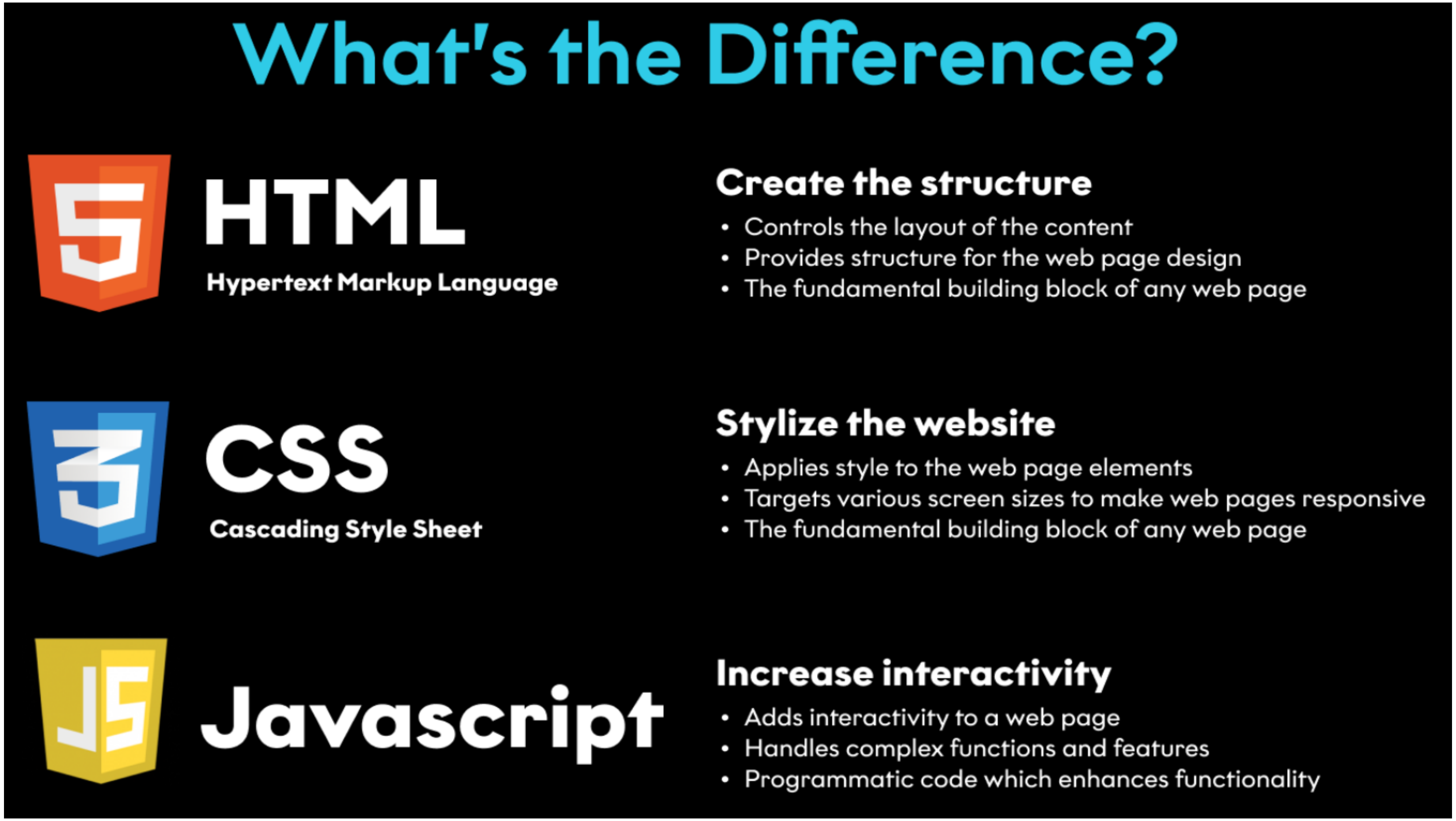
- JavaScript is a client side scripting language (interpreted programming language)
- JavaScript make web pages interactive
- Open source and cross-platform
- Case sensitive
- Most commonly used as a part of web pages
- JS was created to make web pages more Dynamic (Change content on a page directly from inside the browser)
- Supported by all major browsers and enabled by default

Why JavaScript?

- JavaScript adds behavior to web pages
- Show or hide more information with the click of a button
- Change the color of a button when the mouse hovers over it
- Less server interaction
- Immediate feedback to the visitors

Less Server Interaction
Immediate feedback to the visitors
Increased Interactivity
Richer Interfaces

HTML, CSS & JavaScript



What's the Difference?

HTML
Hypertext Markup Language

Create the structure
- Controls the layout of the content
- Provides structure for the web page design
- The fundamental building block of any web page

CSS
Cascading Style Sheet

Stylize the website
- Applies style to the web page elements
- Targets various screen sizes to make web pages responsive
- The fundamental building block of any web page

Javascript

Increase interactivity
- Adds interactivity to a web page
- Handles complex functions and features
- Programmatic code which enhances functionality

How to Add JavaScript

❖ **Internal JS** - Internal JavaScript code is code that's placed anywhere within the web page between the HTML tags

```
<script>
 alert("Happy Learning");
</script>
```

❖ **External JS**

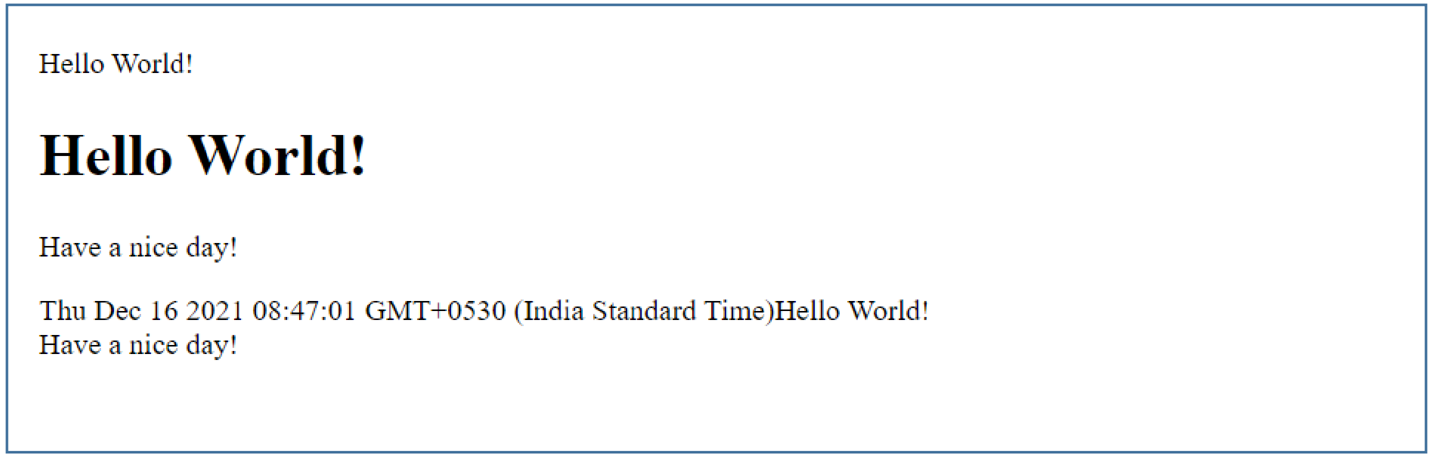
- JavaScript code placed in a file separate from the HTML code is called external Javascript.
- External JavaScript code is written and used in the same way as internal Javascript.
- The file should have the ".js" extension.

```
<script src="myScript.js"></script>
```

Interact with HTML using JS

- **Using Document Object**

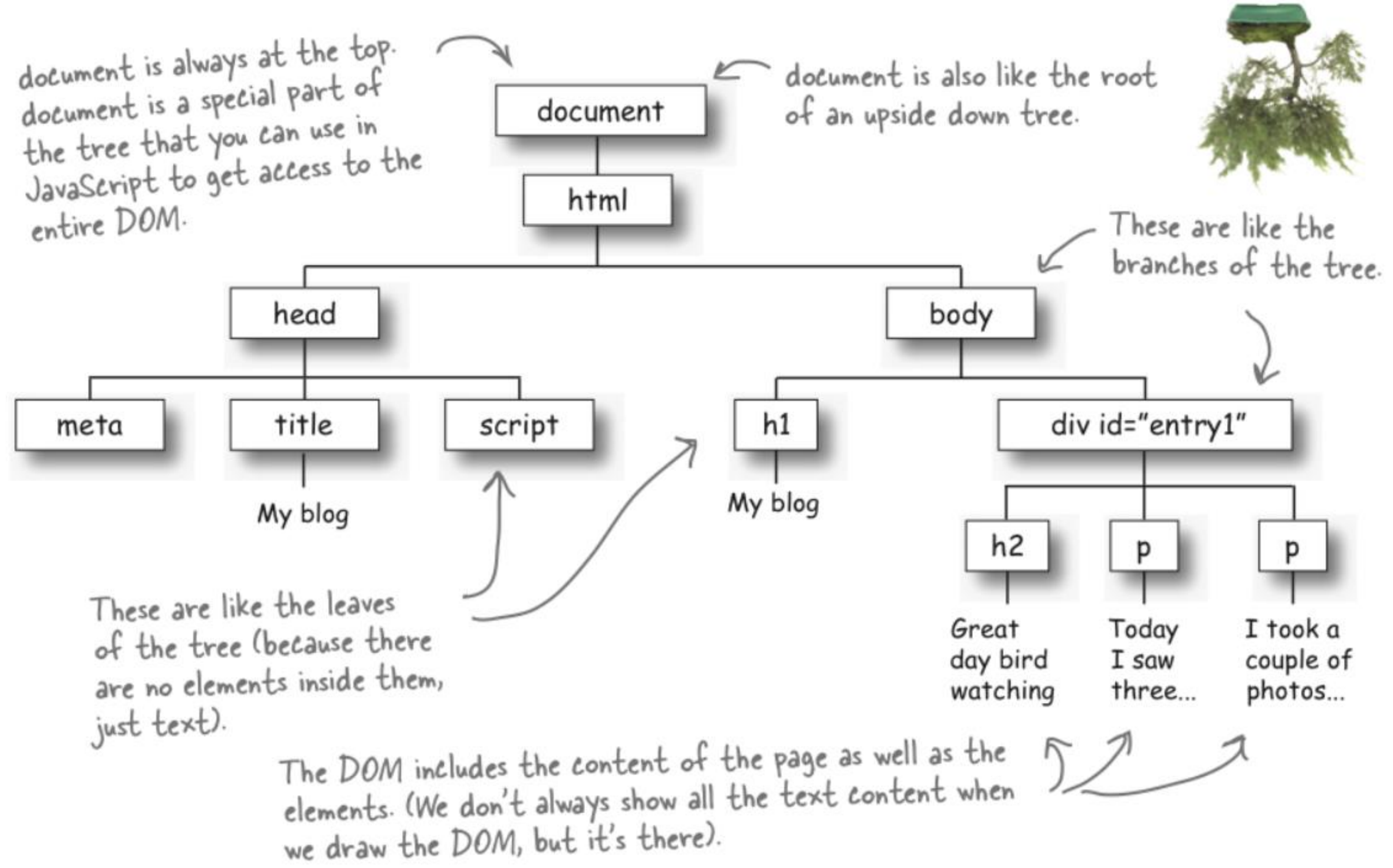
```
document.write("Hello World!");
document.write("<h1>Hello World!</h1><p>Have a nice day!</p>");
document.write(Date());
document.write("Hello World! <br>");
document.write("Have a nice day!");
```



```
Hello World!

Hello World!

Have a nice day!

Thu Dec 16 2021 08:47:01 GMT+0530 (India Standard Time)Hello World!
Have a nice day!
```

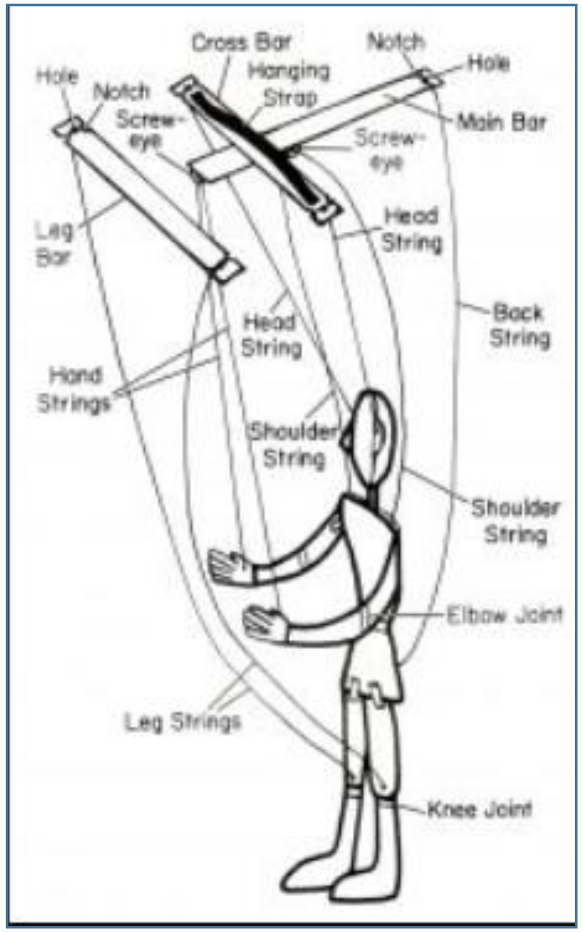
What is DOM ?

- When a web page is loaded, the browser creates a **Document Object Model** of the page
- The **HTML DOM** model is constructed as a tree of **Objects**:


document is always at the top. document is a special part of the tree that you can use in JavaScript to get access to the entire DOM.
document
document is also like the root of an upside down tree.
html
These are like the branches of the tree.
head
body
meta
title
script
h1
div id="entry1"
My blog
My blog
h2
p
p
Great day bird watching
Today I saw three...
I took a couple of photos...
These are like the leaves of the tree (because there are no elements inside them, just text).
The DOM includes the content of the page as well as the elements. (We don't always show all the text content when we draw the DOM, but it's there).

DOM continue..

- **HTML page as a puppet**, then the **DOM will be the strings** and the points at which they attach to the puppet



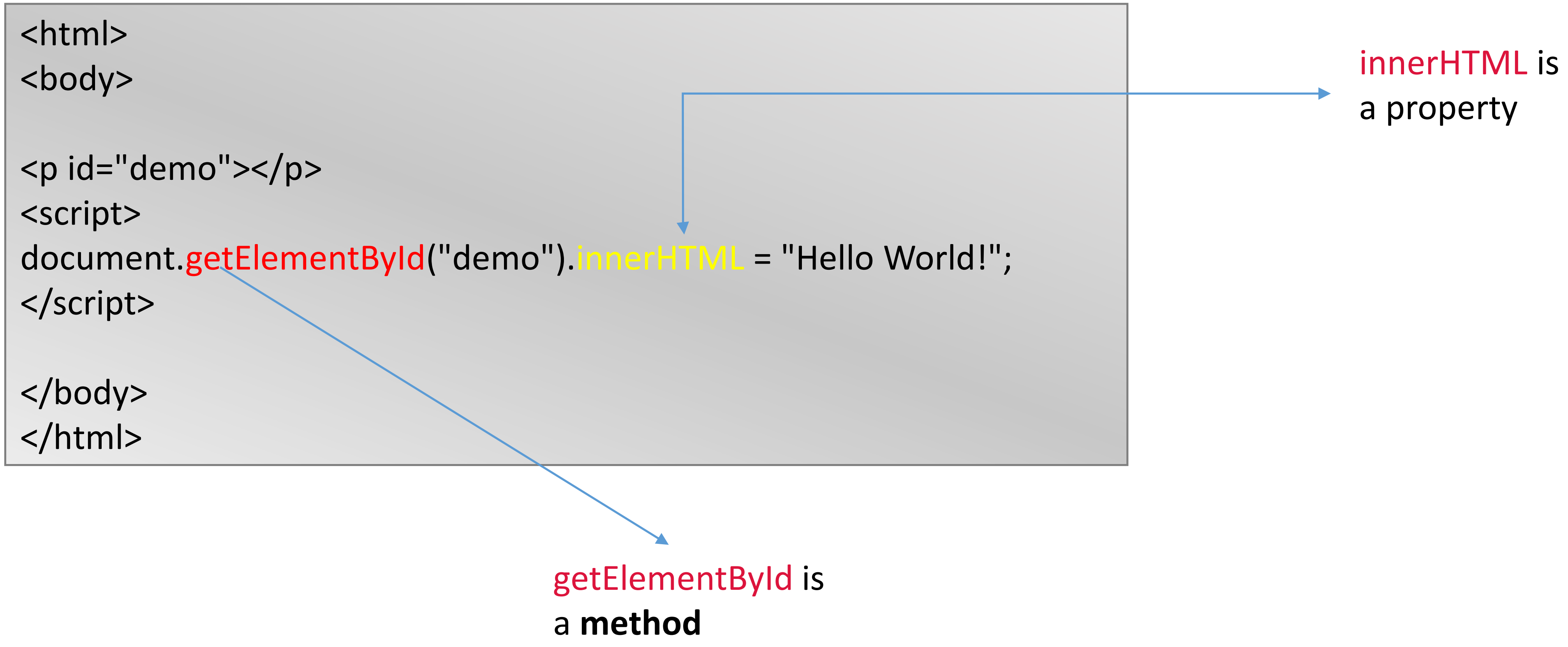
What we can do using DOM?

Using DOM, JavaScript gets all the power it needs to create/update HTML:

- JavaScript can **change all the HTML elements** in the page
- JavaScript can **change all the HTML attributes** in the page
- JavaScript can **change all the CSS styles** in the page
- JavaScript can **remove existing HTML elements and attributes**
- JavaScript can **add new HTML elements and attributes**
- JavaScript can **react to all existing HTML events** in the page

DOM Methods

- HTML **DOM methods are actions** you can perform (on HTML Elements)
- HTML **DOM properties are values** (of HTML Elements) that you can set or change



```
<html>
<body>

<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

DOM Methods

- **getElementById -** To change/modify the content of an HTML element

```
<script>
document.getElementById("p1").innerHTML = "New text!";
</script>
```

→ Changing HTML Content

```
<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>
```

→ Changing the Value of an Attribute

```
<script>
  document.getElementById("myP").style.backgroundColor = "red";
</script>
```

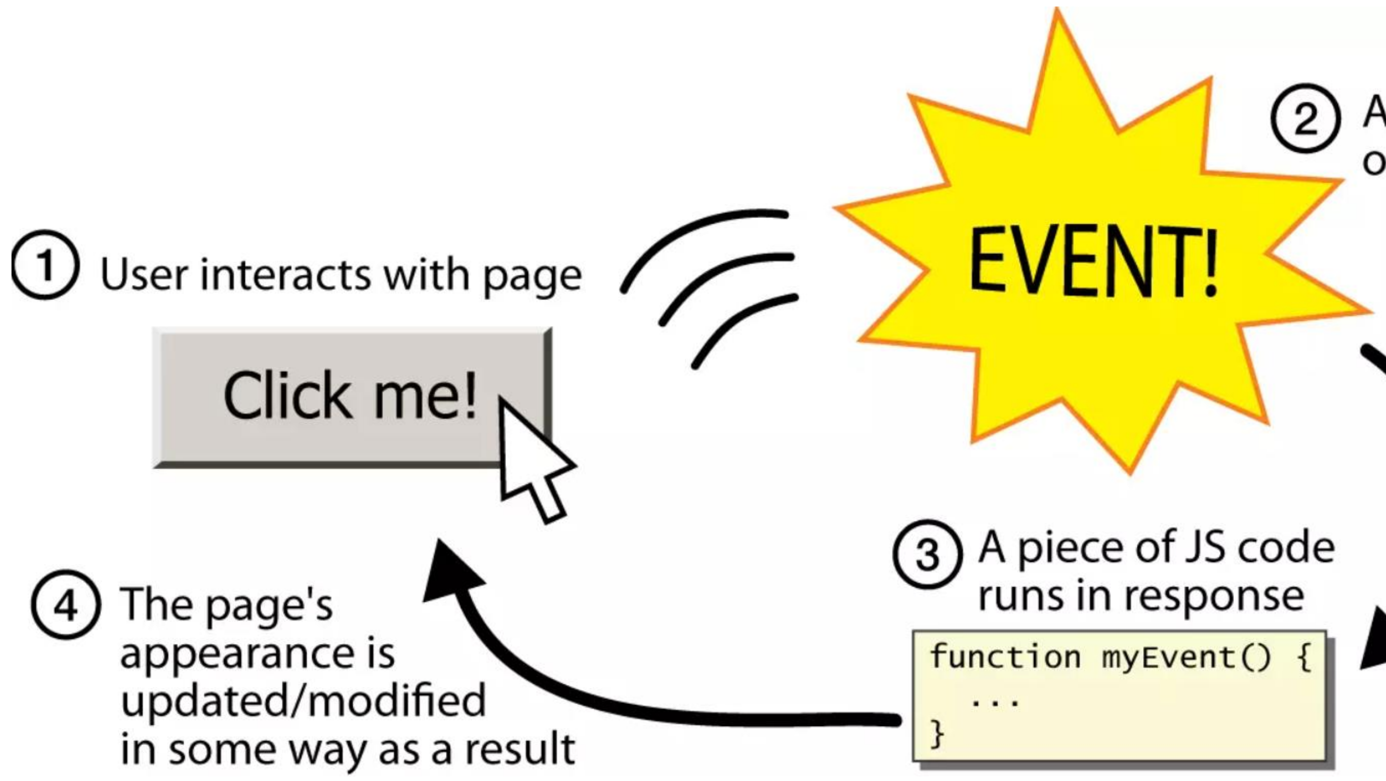
→ Changing Style property

Events in JavaScript

An HTML event can be something the browser does, or something a user does

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked



① User interacts with page
Click me!
EVENT!
② An "event" occurs
③ A piece of JS code runs in response
④ The page's appearance is updated/modified in some way as a result

```
function myEvent() {
   ...
}
```

Events in JavaScript

CREDO SYSTEMZ
simplifying IT

Common HTML Events:

- **Onclick** - The user clicks an HTML element
- **Onchange** - An HTML element has been changed
- **Onmouseover** - The user moves the mouse over an HTML element
- **Onmouseout** - The user moves the mouse away from an HTML element
- **Onkeypress -** The event occurs when the user presses a key

Events in JavaScript

Common HTML Events:

- **Onclick** - The user clicks an HTML element

```
<div>
  <p class="name">Learning JS events</p>
  <button id="btn" onclick="changeColor()">Change to Blue</button>
</div>
```

```
function changeColor() {
   document.getElementById("btn").style.color = "blue";
}
```

- **Onchange** - An HTML element has been changed

```
<select id="cities" name="city" onchange="selectCity()">
          <option value="chn">Chennai</option>
          <option value="blr">Bengaluru</option>
          <option value="hyd" selected>Hyderabad</option>
      </select>
<p id="slcCty"></p>
```

```
function selectCity() {
            var cty = document.getElementById("cities").value;
            document.getElementById("slcCty").innerHTML
="Selected City: " + cty;
      }
```

Events in JavaScript

CREDO SYSTEMZ
simplifying IT

Common HTML Events:

- **Onmouseover** - The user moves the mouse over an HTML element

```
<div>
  <p class="name">Learning JS events</p>
  <button id="btn" onmouseover="changeBGColor()">Change to
Blue</button>
</div>
```

```
function changeBGColor(){
document.getElementById("btn").style.backgroundColor=
"red";
      }
```

- **Onmouseout** - The user moves the mouse away from an HTML element

```
<div>
  <p class="name">Learning JS events</p>
  <button id="btn" onmouseout="changeDefaultColor()">Change to
Blue</button>
</div>
```

```
function changeDefaultColor(){
document.getElementById("btn").style.backgroundColor=
"grey";
      }
```

- **Onkeypress** - The event occurs when the user presses a key

```
<input id="txtBox" type="text" value="" onkeypress="keyPressUpd()">
```

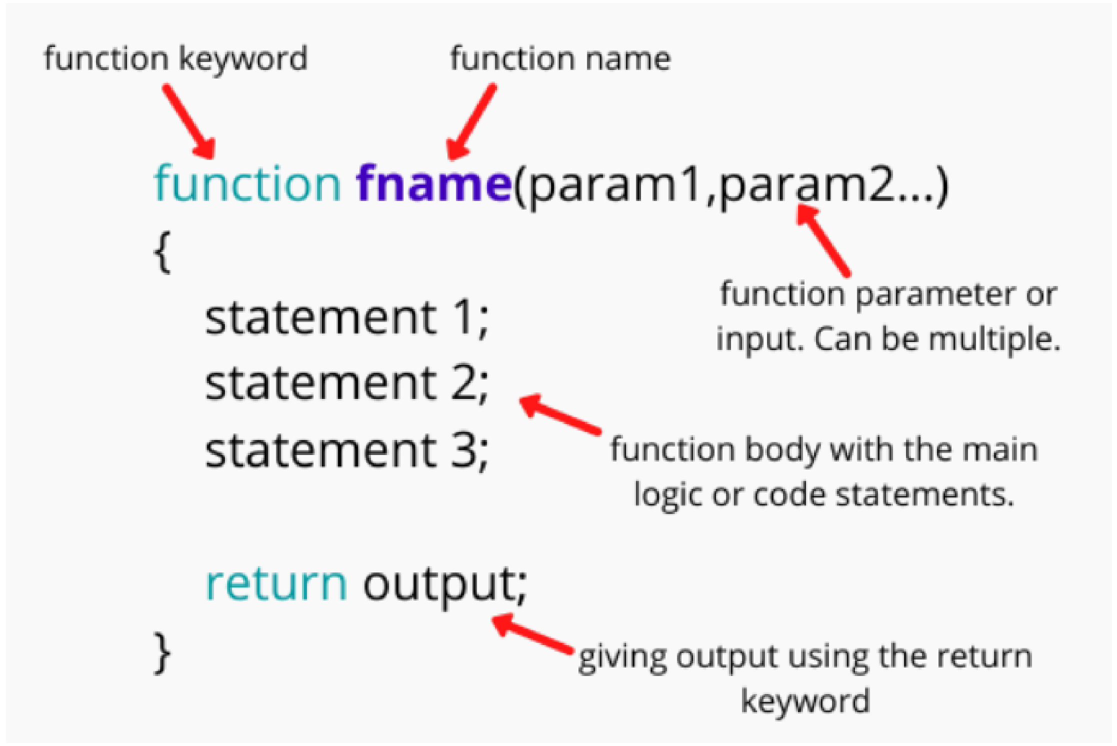
```
function keyPressUpd(){
document.getElementById("txtBox").style.backgroundCol
or= "grey";
      }
```

Function in JS

- A function is a block of code that performs a specific task.

```
//defining a function
function <function-name>()
{
   // code to be executed
};

//calling a function
<function-name>();
```

```
//defining a function
function ShowMessage() {
   alert("Hello World!");
}

//calling a function
ShowMessage();
```



```
function fname(param1,param2...)
{
    statement 1;
    statement 2;
    statement 3;

    return output;
}
```

function keyword
function name
function parameter or input. Can be multiple.
function body with the main logic or code statements.
giving output using the return keyword

Function in JS

Function with parameters:

```
//defining a function
function ShowMessage(firstName, lastName) {
   alert("Hello " + firstName + " " + lastName);
}

//calling a function
ShowMessage(“Ashok", “Kumar");
```

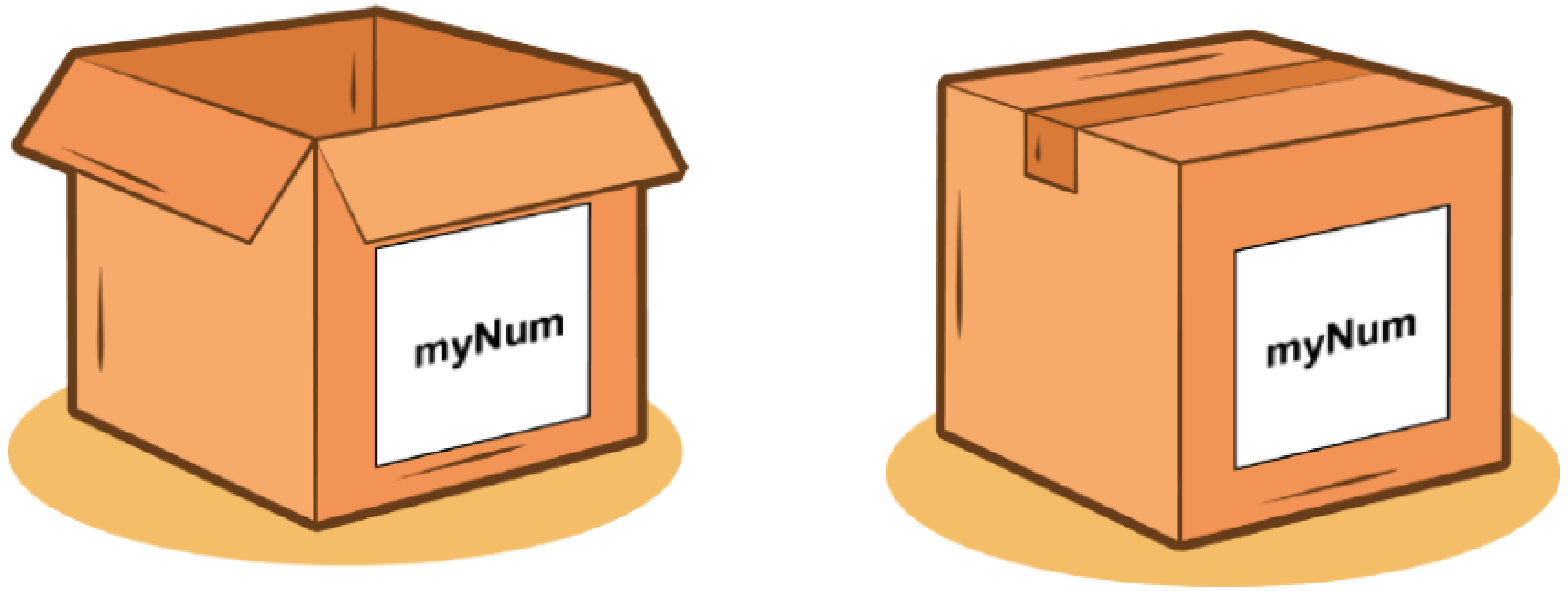
Function with Return Value:

```
//defining a function
function Sum(val1, val2) {
   return val1 + val2;
};

//calling a function
var result = Sum(10,20); // returns 30
```

Variables in JS

- Variables are containers for storing data (values) that hold information and allow us access them later
- All JavaScript variables must be identified with unique names
- These unique names are called identifiers
- We will think of this as a box that has a label on it.

```
var myNum;
```



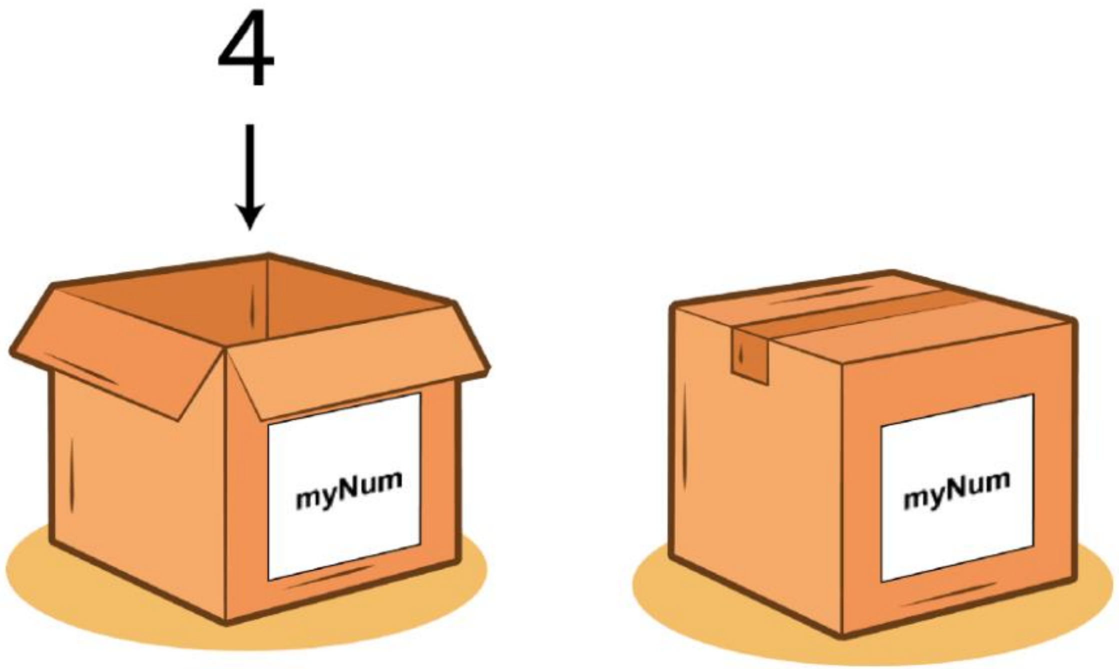
Variables in JS

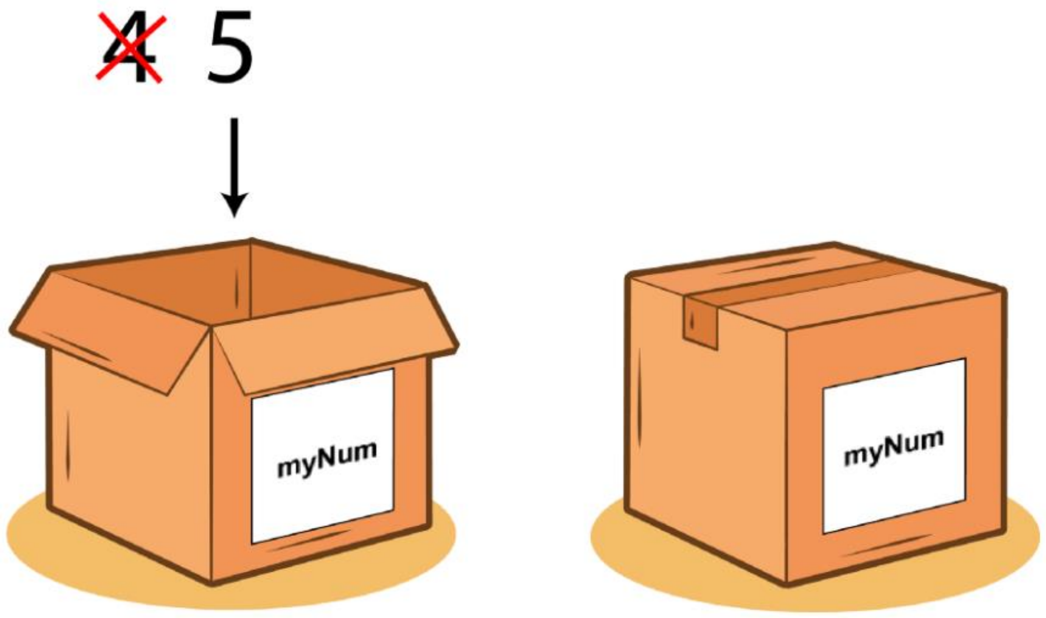
- We can visualise this a box that has a value added to it. Below we add 4 to myNum variable

```
var myNum = 4;
```

```
var myNum = 4
myNum = 5
```

Changing variables after initialisation (reassignment)





Variables Scopes

JavaScript has 3 types of scope

- **Block scope** - Variables declared inside a { } block **cannot be accessed** from outside the block
- **Function scope** - Variables declared within a JavaScript function, become **LOCAL** to the function
- **Global scope** - A variable declared outside a function, becomes **GLOBAL**

```
{
  let x = 2;
}
// x can NOT be used here
```

Variables declared with the var keyword can NOT have block scope.

```
{
  var x = 2;
}
// x CAN be used here
```

```
// code here can NOT use myName

function myFunction() {
  let myName = "Raja";
  // code here CAN use myName
}

// code here can NOT use myName
```

```
let myName = "Ram";
// code here can use myName

function myFunction() {
// code here can also use myName
}
```

Declare Variable

Three ways to declare a variable in javaScript

- Using var

```
var x = 5;
var y = 6;
var z = x + y;
```

- Using let

```
let x = "John Doe";
```

- Using const

```
const PI = 3.141592653589793;
```

Data Types in JS

- A data type specifies the type of data that a variable can store
- JavaScript provides different data types to hold different types of values

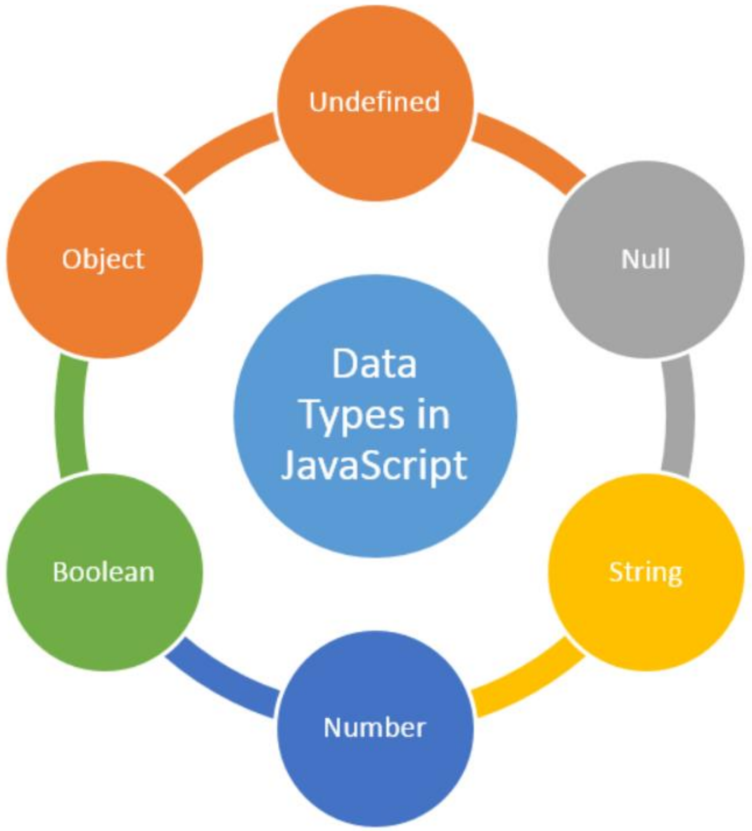
Two types of data types

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

```
var a= 10; //holding number
var b= "Ram"; //holding string
```



Primitive Data types

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	a data type whose variable is not initialized e.g. let a; you can say that undefined means lack of value or unknown value
Null	represents null i.e. no value at all

```
var b= "Ram"; //holding string
var a= 10; //holding number
Console.log(Boolean(10 > 9)) // boolean
var myVar = null;
```

Primitive Data types

- null and undefined are primitive values in JavaScript.
- A null value means absence.
- An undefined value means lack of value.
- A null or undefined value evalutes to false in conditional expression.

Non-Primitive Data types

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values

```
let person = {
   firstName: 'John',
   lastName: 'Doe'
};
```

```
var stringArray = [“John", “Doe"];

var numericArray = [1, 2, 3, 4];
```

JavaScript Operators

An operator performs some operation on single or multiple operands and produces a result

<Left operand> operator <right operand>

For example, in 1 + 2,
+ sign is an operator and
1 is left side operand and
2 is right side operand

Types of Operators

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators
4. Assignment Operators
5. Conditional Operators
6. Ternary Operator

JavaScript Operators - Arithmetic

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

```
var x = 5, y = 10;

var 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

var 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
```

JavaScript Operators - Comparison

CREDO SYSTEMZ
simplifying IT

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: Comparison Operators

```
var a = 5, b = 10, c = "5";
var 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
```

JavaScript Operators - Logical

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

```
var 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
```

JavaScript Operators - Ternary

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

Syntax:

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

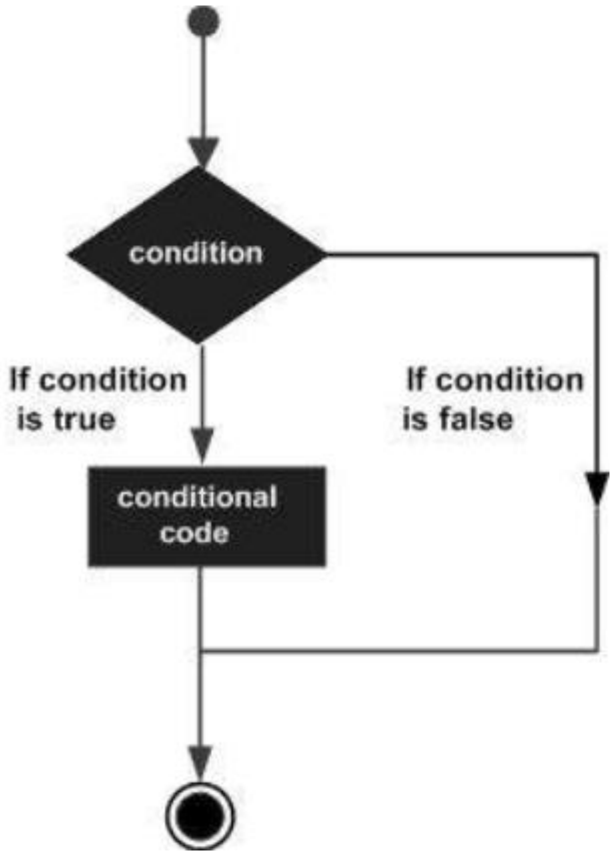
Example:

```
var a = 10, b = 5;

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

JavaScript conditional statements

JavaScript includes if-else conditional statements to control the program flow, similar to other programming languages.

- if condition
- if-else condition
- else if condition



Conditional statements – if condition

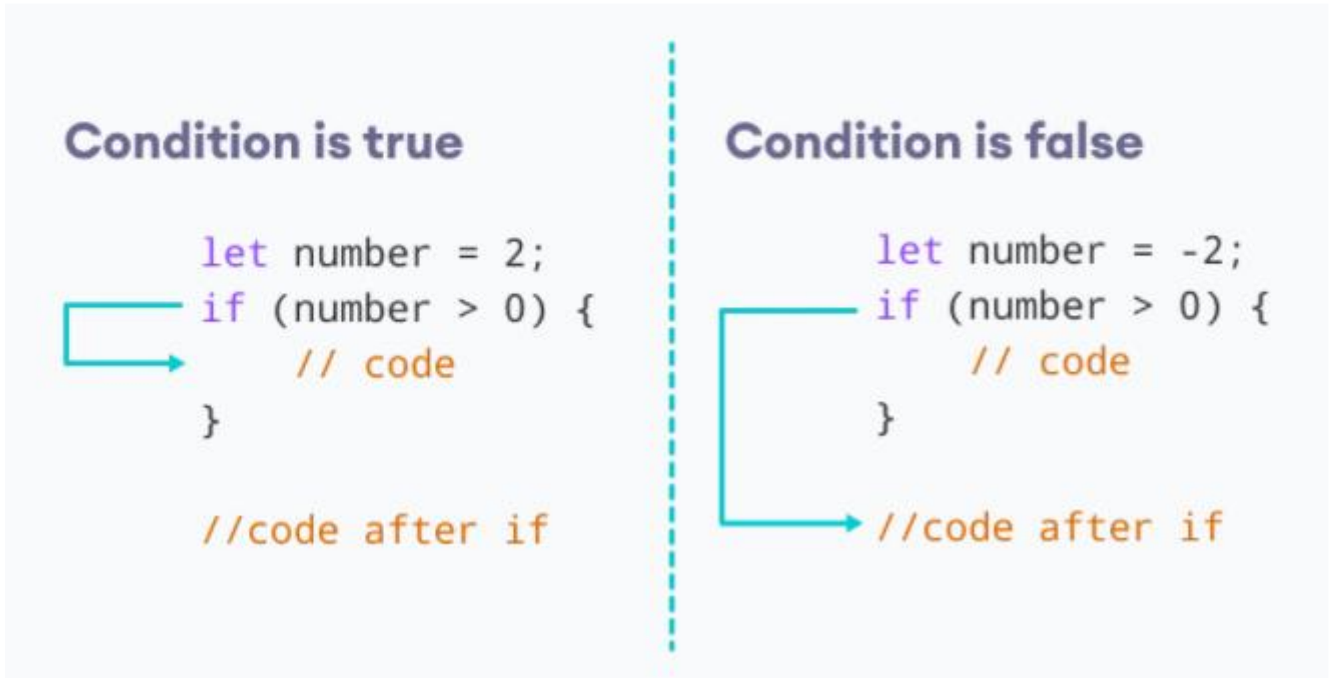
- The keyword if tells JavaScript to start the conditional statement.
- (1 > 0) is the condition to test, which in this case is true — 1 is greater than 0.
- The part contained inside curly braces {} is the block of code to run.
- Because the condition passes, the variable outcome is assigned the value "if block".

Syntax:

```
if(condition expression)
{
   // code to be executed if condition is true
}
```

Example:

```
if( 1 > 0)
{
   alert("1 is greater than 0");
}
if( 1 < 0)
{
   alert("1 is less than 0");
}
```



Condition is true

```
let number = 2;
if (number > 0) {
    // code
}

//code after if
```

Condition is false

```
let number = -2;
if (number > 0) {
    // code
}

//code after if
```

Conditional statements - Else condition

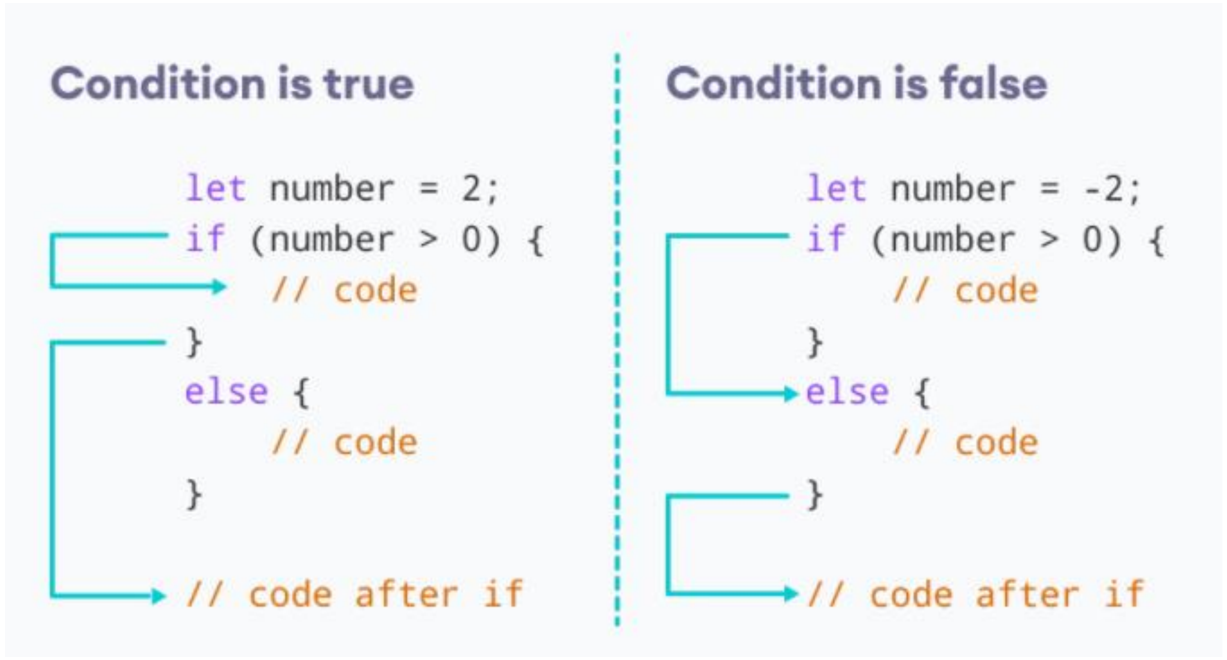
Use else statement when you want to execute the code every time when if condition evaluates to false.

Syntax:

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

Example:

```
let num = 2
// let num = -2
if(number > 0)
{
   alert(num +" is greater than 0");
}
else
{
   alert(num +" is less than 0");
}
```



```
let number = 2;
if (number > 0) {
    // code
}
else {
    // code
}

// code after if
```

```
let number = -2;
if (number > 0) {
    // code
}
else {
    // code
}

// code after if
```

Conditional statements – Else if condition

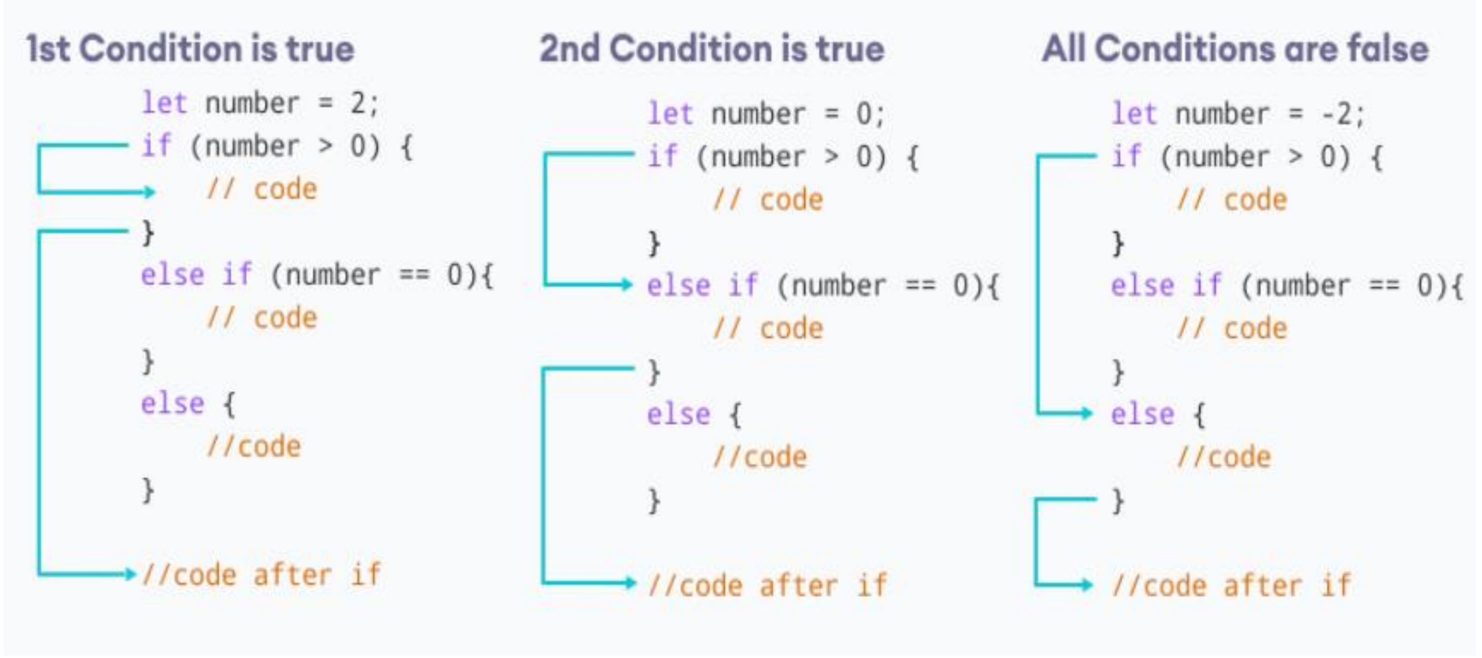
Use "else if" condition when you want to apply second level condition after if statement

Syntax:

```
if(condition expression)
{
   //Execute this code block
}
else if(condition expression){
   //Execute this code block
}
```

Example:

```
// check if the number if positive, negative or zero
let number = 2;
number is positive");
} if (number > 0) {
   console.log("The
else if (number == 0) {
  console.log("The number is 0");
}
else {
   console.log("The number is negative");
}
```



```
let number = 2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

```
let number = 0;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

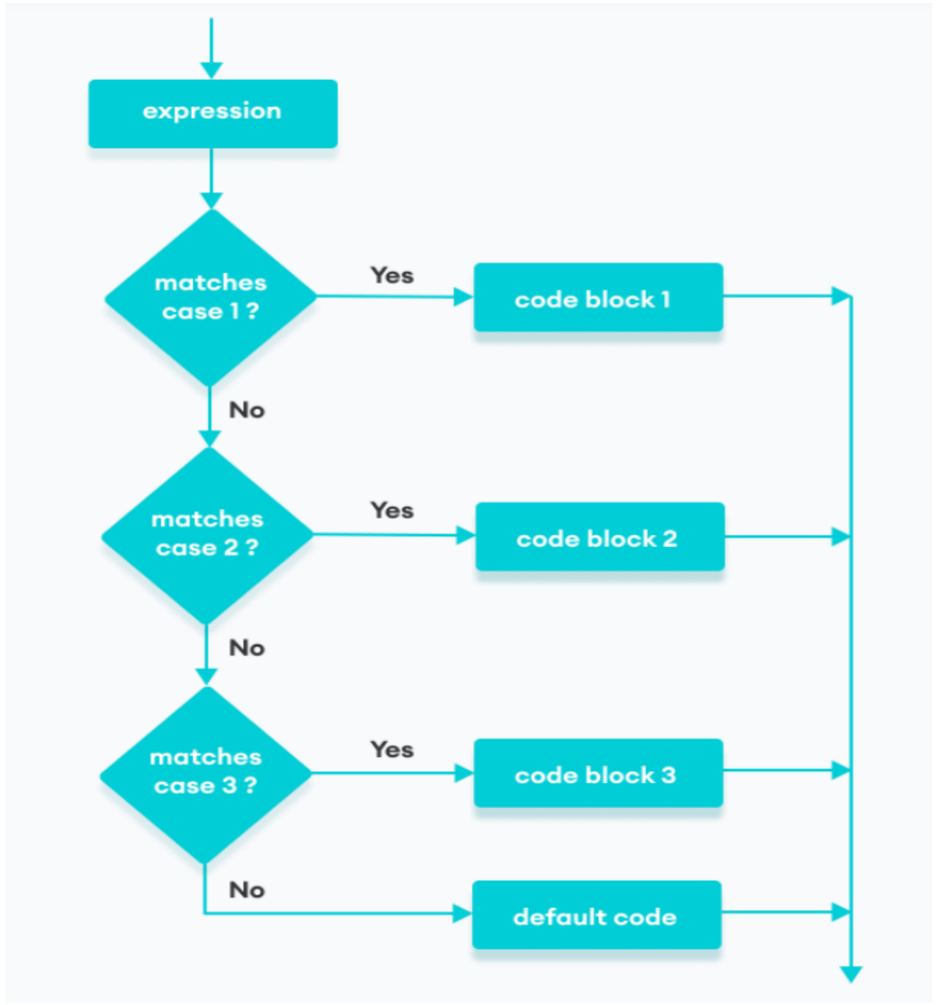
```
let number = -2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

JavaScript conditional statements - Switch

- The JavaScript switch statement is used in decision making.
- The switch statement evaluates an expression and executes the corresponding body that matches the expression's result.

Syntax:

```
switch(expression or literal value){
   case 1:
      //code to be executed
     break;
   case 2:
      //code to be executed
      break;
   case n:
      //code to be executed
      break;
   default:
      //default code to be executed
      //if none of the above case executed
}
```



JavaScript conditional statements - Switch

Example:

```
let a = 2;

switch (a) {

   case 1:
      a = 'one';
      break;
   case 2:
      a = 'two';
      break;
   default:
      a = 'not found';
      break;
}
console.log(`The value is ‘+{a});
```

Example:

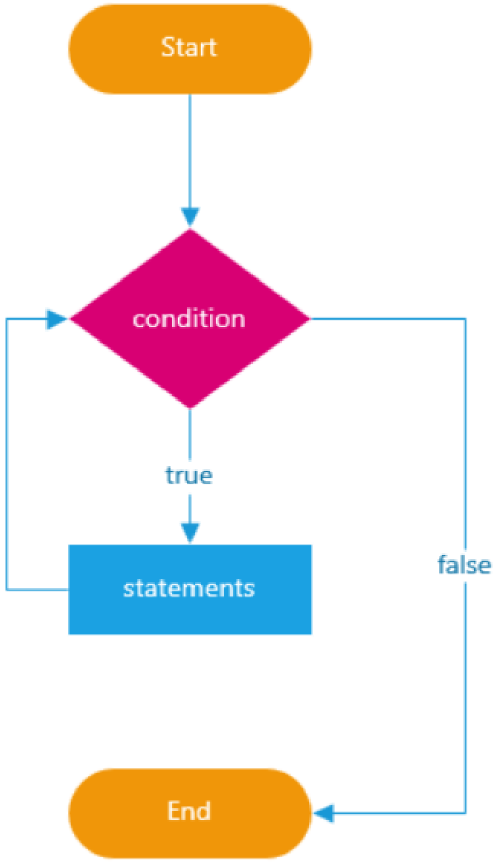
```
let str = "bill";

switch (str)
{
   case "steve":
      alert("This is Steve");
   case "bill":
      alert("This is Bill");
      break;
   case "john":
      alert("This is John");
      break;
   default:
      alert("Unknown Person");
      break;
}
```

JavaScript Loops

In programming, loops are used to repeat a block of code

For example, if you want to show a message 100 times, then you can use a loop. It's just a simple example; you can achieve much more with loops.

- For loop
- While loop
- do while



JavaScript Loops - for

The for loop requires following three parts.

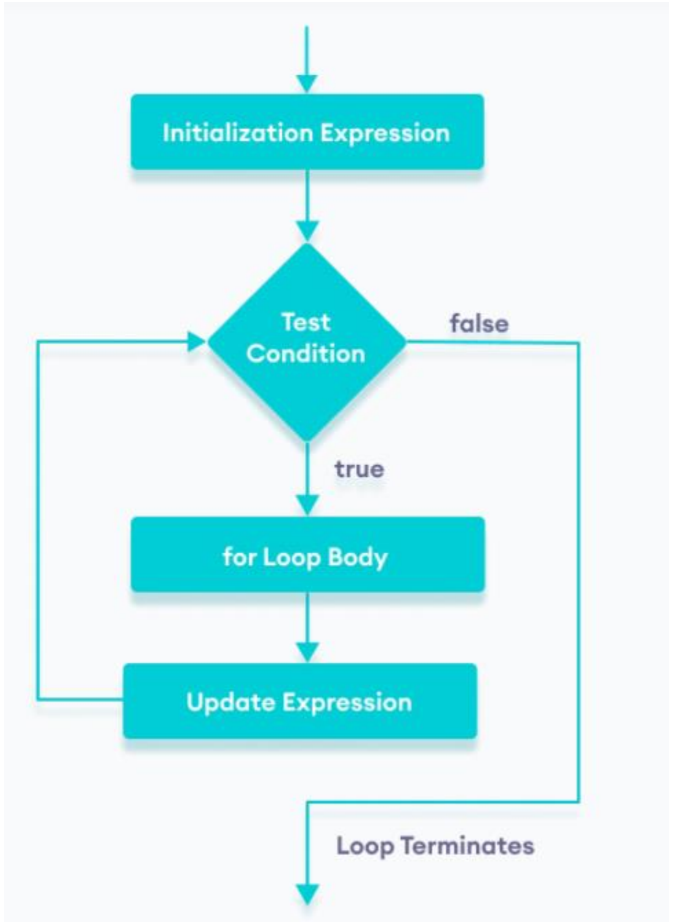
- **Initializer**: Initialize a counter variable to start with
- **Condition**: specify a condition that must evaluate to true for next iteration
- **Iteration**: increase or decrease counter

Syntax

```
for(initializer; condition; iteration)
{
   // Code to be executed
}
```

Example

```
for (var i = 0; i < 5; i++)
{
   console.log(i);
}

let n = 5;

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



JavaScript Loops - While

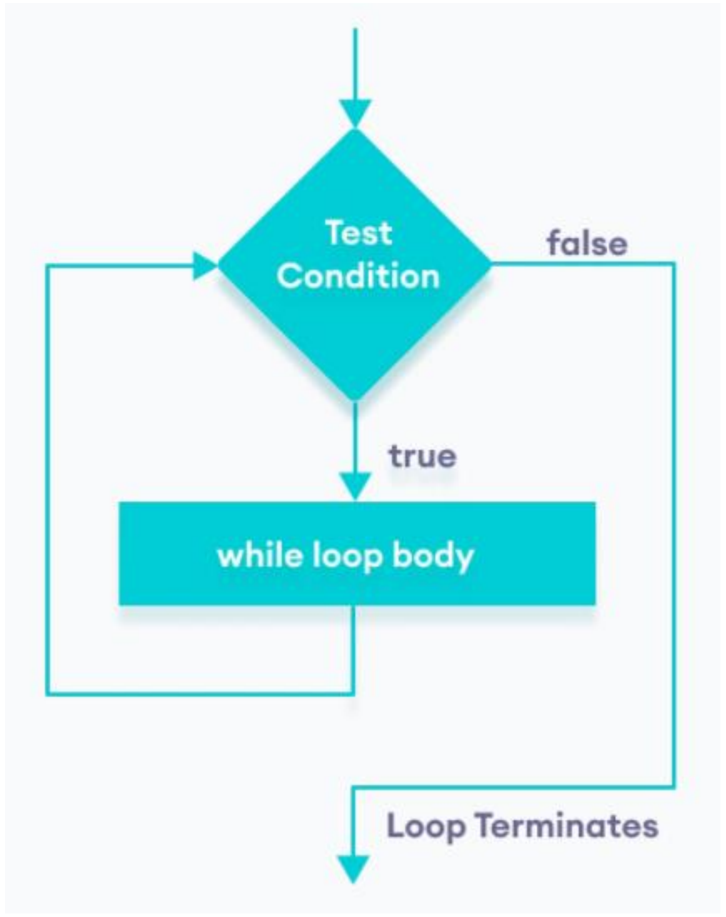
- A while loop evaluates the condition inside the parenthesis ()
- If the condition evaluates to true, the code inside the while loop is executed.
- The condition is evaluated again.
- This process continues until the condition is false.
- When the condition evaluates to false, the loop stops.

Syntax

```
while(condition expression)
{
   /* code to be executed
   till the specified condition is true */
}
```

Example

```
var i =0;

while(i < 5)
{
   console.log(i);
   i++;
}
```



JavaScript Loops – do While

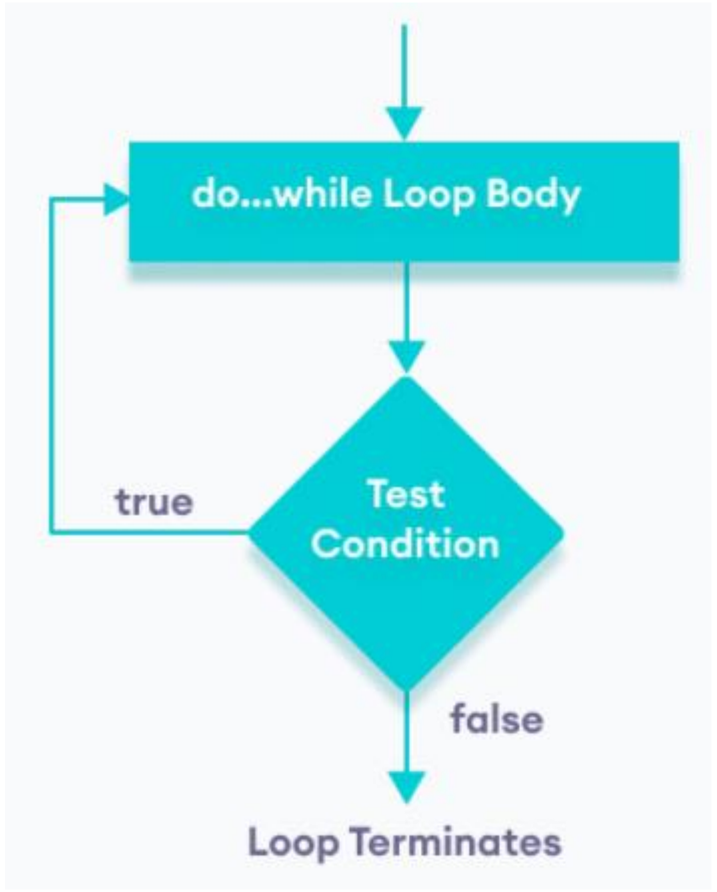
- The body of the loop is executed at first. Then the **condition** is evaluated
- do...while loop is similar to the while loop. The only difference is that in do…while loop, the body of loop is executed at least once

Syntax

```
do{
   //code to be executed
}while(condition expression)
```

Example

```
// program to display numbers
let i = 1;
const n = 5;

// do...while loop from 1 to 5
do {
   console.log(i);
   i++;
} while(i <= n);
```



String Methods

- **length**: Returns the number of characters in a string
- **replace():** method replaces a specified value with another value in a string
- **toUpperCase()**: A string is converted to upper case
- **toLowerCase():** A string is converted to lower case
- **concat():** joins two or more strings, can be used instead of the plus operator
- **trim():** method removes whitespace from both sides of a string
- **charAt():** method returns the character at a specified index (position) in a string
- **indexOf():** method returns the position of the first occurrence of a specified value in a string, method returns -1 if the value is not found
- **Slice():** extracts a part of a string and returns the extracted part in a new string, method takes 2 parameters: the start position, and the end position
- **Substring():** is similar to slice(), difference is that substring() cannot accept negative indexes. omit the second parameter, substring() will slice out the rest of the string
- **split():** method divides a String into an ordered list of substrings and returns them as an array

String Methods

Example

```
let txt = “JavaScript";
let length = txt.length;

let text = “I am learning JavaScript";
let newText = text.replace(“learning", “working");

let text = "Please visit Madras, I love Madras!";
let newText = text.replace(/Madras/g, “Chennai");

let text1 = "Hello World!";
let text2 = text1.toUpperCase();

let text1 = "Hello World!";      // String
let text2 = text1.toLowerCase();

let text1 = "Hello";
let text2 = "World";
let text3 = text1.concat(" ", text2);

let text1 = "      Hello World!      ";
let text2 = text1.trim();
```

Example

```
let text = "HELLO WORLD";
let char = text.charAt(0);

const message = "JavaScript is fun";

// check the first occurrence of 'i' in message
let result = message.indexOf("i");
console.log(result);

str.indexOf(searchValue, fromIndex)

let str = "Apple, Banana, Kiwi";
let part = str.slice(7, 13);

let str = "Apple, Banana, Kiwi";
let part = str.substring(7, 13);
let part1 = str.substring(7);

let message = "JavaScript is fun";
let result = message.split(" ");
```

Array

An array is an object that can store multiple values, create an array is by using an array literal []

Create Array:

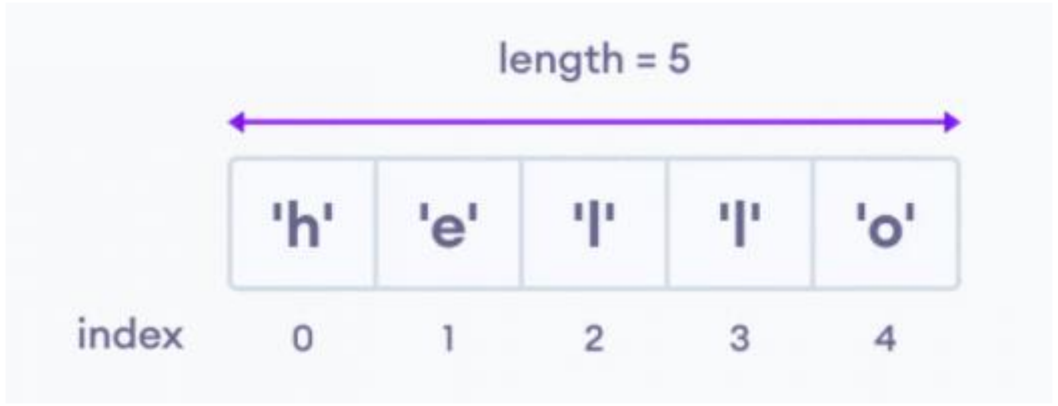
Example:

```
// empty array
const myList = [ ];

// array of numbers
const numberArray = [ 2, 4, 6, 8];

// array of strings
const stringArray = [ 'eat', 'work', 'sleep'];

// array with mixed data types
const newData = ['work', 'exercise', 1, true];
```

Access Elements of an Array:

Example:

```
const myArray = ['h', 'e', 'l', 'l', 'o'];

// first element
console.log(myArray[0]);  // "h“

// second element
console.log(myArray[1]); // "e"
```



Array Methods

CREDO SYSTEMZ simplifying IT

An array is an object that can store multiple values, create an array is by using an array literal []

Arrays to Strings:

```
const elec = [“Mobile", “Ipad", “Laptop", “Chargers"];
console.log(elec.toString())
```

join():

```
const elec = [“Mobile", “Ipad", “Laptop", “Chargers"];
console.log(elec.join("-"))
//mobile-ipad-laptop-charges
```

Add an Element to an Array:

push() and unshift() to add elements to an array

```
let dailyActivities = ['eat', 'sleep'];

// add an element at the end
dailyActivities.push('exercise');

console.log(dailyActivities); //  ['eat', 'sleep', 'exercise']
```

```
let dailyActivities = ['eat', 'sleep'];

//add an element at the start
dailyActivities.unshift('work');

console.log(dailyActivities); // ['work', 'eat', 'sleep']
```

Array Methods

CREDO SYSTEMZ
simplifying IT

Change the Elements of an Array:

Add elements or Change the elements by accessing the index value

```
let dailyActivities = [ 'eat', 'sleep'];

// this will add the new element 'exercise' at the 2 index
dailyActivities[2] = 'exercise';

console.log(dailyActivities); // ['eat', 'sleep', 'exercise']
```

Remove an Element from an Array:

pop() method to remove the last element from an array. The pop() method also returns the removed element

```
let dailyActivities = ['work', 'eat', 'sleep', 'exercise'];

// remove the last element
dailyActivities.pop();
console.log(dailyActivities); // ['work', 'eat', 'sleep']

// remove the last element from ['work', 'eat', 'sleep']
const removedElement = dailyActivities.pop();
```

shift() method removes the first element and also returns the removed element

```
let dailyActivities = ['work', 'eat', 'sleep'];

// remove the first element
dailyActivities.shift();

console.log(dailyActivities); // ['eat', 'sleep']
```

Array Methods

Array Length:

Number of elements in an array using the length property

```
const dailyActivities = [ 'eat', 'sleep'];

// this gives the total number of elements in an array
console.log(dailyActivities.length); // 2
```

Concat:

The concat() method creates a new array by merging existing array

```
const myDailyAct = [ 'eat', 'sleep'];
const myAddiction = [“work", “play", “roam"];

const myChildren = myDailyAct.concat(myAddiction);
```

splice():

- The splice() method can be used to add new items to an array
- The first parameter (2) defines the position **where** new elements should be **added**.
- The second parameter (0) defines **how many** elements should be **removed**.
- The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");

const remItems = fruits.splice(2, 2, "Lemon", "Kiwi");

// to remove items
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);
```

Array Methods

slice():

- The slice() method slices out a piece of an array into a new array
- The slice() method does not remove any elements from the source array.

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(2);

//slice() method can take two arguments like slice(1, 3)
const citrus = fruits.slice(1,4);
```

Sorting an Array:

The sort() method sorts an array alphabetically

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
```

Reversing an Array:

The reverse() method reverses the elements in an array

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.reverse();
```

Array Methods

indexOf():

searches an element of an array and returns its position

```
//finding the index position of string
const position = dailyActivities.indexOf('work');
console.log(position); // 2
```

includes():

The includes() method returns true if an array contains a specified value

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.includes("Mango");
```

isArray():

Check if an object is an array

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let result = Array.isArray(fruits); // true

let text = “testStringArray";
let result = Array.isArray(text); // false
```

Array Loops – For & Foreach

for:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}
```

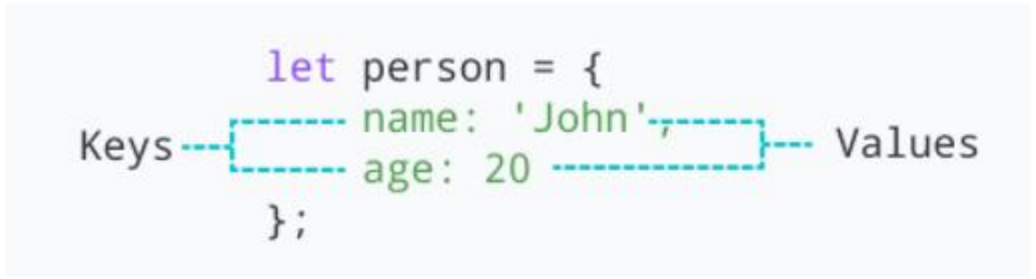
foreach():

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.forEach(function(item){
  console.log(item);
})
```

Object

JavaScript object is a non-primitive data-type that allows you to store multiple collections of data

Syntax:
```
const object_name = {
  key1: value1,
  key2: value2
}
```



```
let person = {
    name: 'John',
    age: 20
};
```

```
// object creation
let person = {
   name: 'John',
   age: 20
};
console.log(typeof person); // object
```

"key: value" pairs are called **properties**

Object

Accessing Object Properties:

Syntax:
objectName.key
Or
objectName["propertyName"]

```
const person = {
   name: 'John',
   age: 20,
};

// accessing property
console.log(person.name); // John

const person = {
   name: 'John',
   age: 20,
};

// accessing property
console.log(person["name"]); // John
```

Object – for...in

Loop through an object Using for...in

```
const student = {
    name: 'John',
    age: 20,
    hobbies: ['reading', 'games', 'coding'],
};

// using for...in
for (let key in student) {
    let value;

    // get the value
    value = student[key];

    console.log(key + " - " +  value);
}
```

JavaScript strict mode

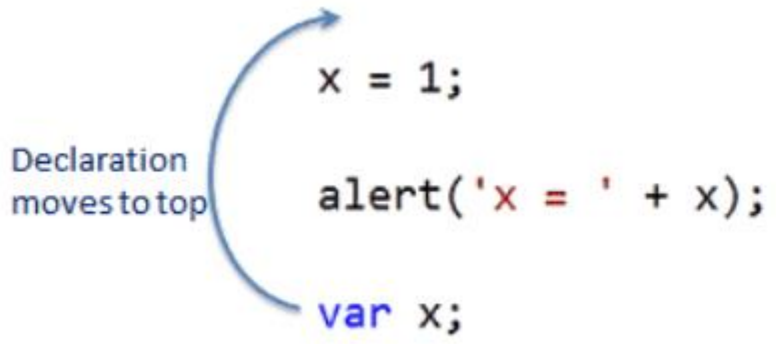
- JavaScript is a loosely typed (dynamic) scripting language
- JavaScript allows strictness of code using "use strict" with ECMAScript 5 or later.
- Write "use strict" at the top of JavaScript code or in a function

```
"use strict";

var x = 1; // valid in strict mode
y = 1; // invalid in strict mode
```

JavaScript Hoisting

- Hoisting is a concept in JavaScript, not a feature. In other scripting or server side languages, variables or functions must be declared before using it
- In JavaScript, variable and function names can be used before declaring it
- The JavaScript compiler moves all the declarations of variables and functions at the top so that there will not be any error. This is called hoisting.

```
x = 1;

alert('x = ' + x); // display x = 1

var x;
```



JavaScript Hoisting