ISWE206L – WEB TECHNOLOGIES

Module:2	Exploring JavaScript	7 hours
JavaScript basics-Variable, Operators, Variable typing, Functions, Global variables, Local variable, The Document Object Model (DOM) – Expressions and Control Flow in JavaScript- Conditional and looping, Explicit casting- JavaScript functions Objects, and Arrays– Regular expression – Form validation- DOM.		

Java Script – Introduction

JavaScript is a lightweight, interpreted **programming** language. It is designed for creating network-centric applications. JavaScript allows you to add interactivity to a web page. It is often used with HTML and CSS to enhance the functionality of a web page such as validating forms, creating interactive maps, and displaying animated charts.

When a web page is loaded i.e. after HTML and CSS have been downloaded, the JavaScript engine in the web browser executes the JavaScript code. The JavaScript code then modifies the HTML and CSS to dynamically update the user interface.

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

Java Script – Introduction

Advantages:

- Less server interaction
- Immediate feedback to the visitors
- Increased interactivity
- Richer interfaces

Limitations:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multi-threading or multiprocessor capabilities.

So that, JavaScript cannot be treated as a full-fledged programming language

HISTORY & NEED

- JAVASCRIPT WAS INVENTED BY BRENDAN EICH AT NETSCAPE (WITH NAVIGATOR 2.0) AND HAS APPEARED IN ALL BROWSERS SINCE 1996.
- THE OFFICIAL STANDARDIZATION WAS ADOPTED BY THE ECMA ORGANIZATION IN 1997
- ECMA-262 IS THE OFFICIAL JAVASCRIPT STANDARD

- JAVASCRIPT WAS DESIGNED TO ADD INTERACTIVITY TO HTML PAGES
- JAVASCRIPT IS A SCRIPTING LANGUAGE.
- A SCRIPTING LANGUAGE IS A LIGHT WEIGHT PROGRAMMING LANGUAGE
- JAVASCRIPT IS USUALLY EMBEDDED DIRECTLY INTO HTML PAGES
- JAVASCRIPT IS AN INTERPRETED LANGUAGE (MEANS THAT SCRIPTS EXECUTE WITHOUT PRELIMINARY COMPILATION)
- EVERYONE CAN USE JAVASCRIPT WITHOUT PURCHASING A LICENSE

Java Script – Introduction

JavaScript can be implemented using JavaScript statements that are placed within the **<script>… </script>** HTML tags in a web page. The syntax is:

```
<script language = "javascript" type = "text/javascript">
                    JavaScript code
                       </script>
```

Language – This attribute specifies what scripting language has used. Typically, its value will be JavaScript.

Type – This attribute is recommended to indicate the scripting language in use and its value should be set to "text/javascript".

Src – This attribute is required when script has to be included as external file.

Java Script – Introduction

- There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript is in <head> section .
- To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file in src attribute as follows:

```
<script type = "text/javascript" src = "filename.js" >
</script>
```

- JavaScript is a case-sensitive language.
- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.

Java Script – Data Types

One of the most fundamental characteristics of a programming language is the set of data types it supports.

JavaScript allows three primitive data types −

- **Numbers,** eg. 123, 120.50 etc.
- **Strings** of text e.g. "This text string" etc.
- **Boolean** e.g. true or false.

Java Script – Variables

- Variables can be thought of as named containers.
- Variable must be declared before using that. Variables are declared with the **var** keyword.
- Storing a value in a variable is called **variable initialization**. Variable initialization can be done at the time of variable creation or at a later point in time when it is needed.
- Should not use any of the JavaScript reserved keywords as a variable name. For example, **break** or **boolean** variable names are not valid.
- 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.
- JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

VARIABLES

- A **JAVASCRIPT VARIABLE** IS SIMPLY A NAME OF STORAGE LOCATION. THERE ARE TWO TYPES OF VARIABLES IN JAVASCRIPT : LOCAL VARIABLE AND GLOBAL VARIABLE.
- RULES FOR DECLARING:
  - NAME MUST START WITH A LETTER (A TO Z OR A TO Z), UNDERSCORE(_), OR DOLLAR($) SIGN.(EX. VARIABLE NAME '**123COLLEGE**' IS INVALID, **_123COLLEGE** IS VALID)
  - AFTER FIRST LETTER WE CAN USE DIGITS (0 TO 9), FOR EXAMPLE VALUE1.
  - JAVASCRIPT VARIABLES ARE CASE SENSITIVE, FOR EXAMPLE X AND X ARE DIFFERENT VARIABLES.

Variables are declared using 'var'

```
Var a= 10;//hold number
Var b="VLR"; //hold string
```

VARIABLES

Local Variables	Global Variables
A JavaScript local variable is declared inside block or function. It is accessible within the function or block only	A variable i.e. declared outside the function or declared with window object is known as global variable.

<script>
function abc(){
var x=10;//local variable
}
</script>
	<script>
	var data=200;//global variable
	function a(){
	document.write(data);
	}
	</script>

DATATYPES

- 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
- JAVASCRIPT PROVIDES DIFFERENT **DATA TYPES** TO HOLD DIFFERENT TYPES OF VALUES. THERE ARE TWO TYPES OF DATA TYPES IN JAVASCRIPT.

Primitive Data Type

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

Non Primitive Data Type

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

PRIMITIVE DATA TYPE

- NUMBERS - A NUMBER CAN BE EITHER AN INTEGER OR A DECIMAL
- STRINGS - A STRING IS A SEQUENCE OF LETTERS OR NUMBERS ENCLOSED IN SINGLE OR DOUBLE QUOTES
- BOOLEAN - TRUE OR FALSE
- UNDEFINED - A VARIABLE WITHOUT A VALUE, HAS THE VALUE UNDEFINED.
- NULL - IN JAVASCRIPT NULL IS "NOTHING".
- IN JAVASCRIPT, THE DATA TYPE OF NULL IS AN OBJECT.

NON PRIMITIVE DATA TYPE

- OBJECT
  - UNORDERED COLLECTION OF PROPERTIES
  - NAME:VALUE PAIRS
  - VAR STUDENT={NAME:"ARUN",REGNO:"21BIT001" ,CGPA:9.2};
  - A = STUDENT.REGNO (OR) A= STUDENT["REGNO"]
- ARRAYS
  - COLLECTION OF VALUES IN A SINGLE VARIABLE
  - VAR FRUITS = ["APPLE","ORANGE","MANGO"] OR VAR NUM=[1,2,3]
- DYNAMIC DATA TYPES – SAME VARIABLE CAN BE USED FOR DIFFERENT TYPES
  - VAR X=5;
  - VAR X="HAI";

TO DISPLAY A SIMPLE STRING:

```
<!doctype html>
 <html>
<head> <title>hello, world!</title> </head>
 <body>
<script type="text/javascript">
 document.write("hello everyone");
</script>
</body>
</html>
```

Output:

```
hello everyone
```

CONCATENATE STRINGS

- **WE CAN CONCATENATE STRINGS AND DISPLAY THEM AS A SINGLE STRING:**
- <!DOCTYPE HTML>
- <HTML>
- <HEAD> <TITLE>HELLO, WORLD!</TITLE>
- </HEAD> <BODY>
- <SCRIPT TYPE="TEXT/JAVASCRIPT">
- DOCUMENT.WRITE("MY ROLL NUMBER IS " +10);
- </SCRIPT> </BODY> </HTML>

Output:

```
My roll number is 10
```

EXPRESSION DEMO

```
<html>
<body>
<script type="text/javascript">
var x;
x=5+5;
document.write(x);
document.write("<br />"); x="5"+"5";
document.write(x);
document.write("<br />"); x=5+"5";
document.write(x);
document.write("<br />"); x="5"+5;
document.write(x);
document.write("<br />");
</script>
<p>The rule is: If you add a  number and a
string, the result  will be a string.</p>
</body>
</html>
```

10
55
55
55

The rule is: If you add a number and a string, the result will be a string.

```
<script type="text/javascript">
document.write("Sum of 5+5= " + (5+5));
</script>
```

POPUP BOXES – PROMPT

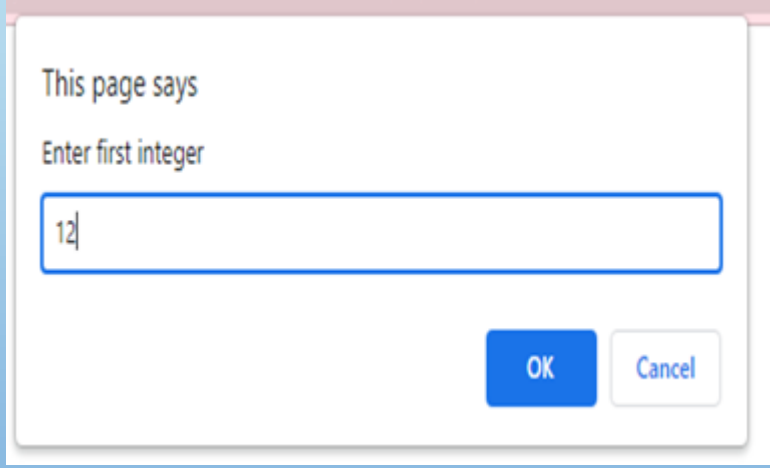
- A PROMPT BOX IS OFTEN USED IF YOU WANT THE USER TO INPUT A VALUE BEFORE ENTERING A PAGE.
- THE USER WILL HAVE TO CLICK EITHER "OK" OR "CANCEL“ AFTER ENTERING AN INPUT VALUE.
- "OK“ - RETURNS THE INPUT VALUE.
- "CANCEL“ - RETURNS NULL.

```
<script>
x=prompt (“Enter ur name”, “ ”)
alert(“Good Morning “+x)
</script>
```

- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

PROMPT

```
<html>
<head>
<title>Using Prompt and Alert Boxes</title>
   <script type = "text/javascript">
      var firstNumber, secondNumber, number1, number2, sum;
            firstNumber =window.prompt( "Enter first integer", "0" );
      secondNumber =window.prompt( "Enter second integer", "0" );
    // convert numbers from strings to integers
    number1 = parseInt( firstNumber );
    number2 = parseInt( secondNumber );
    sum = number1 + number2;
document.writeln( "<h1>The sum is " + sum + "</h1>" );
</script>
</head>
<body></body>
</html>
```



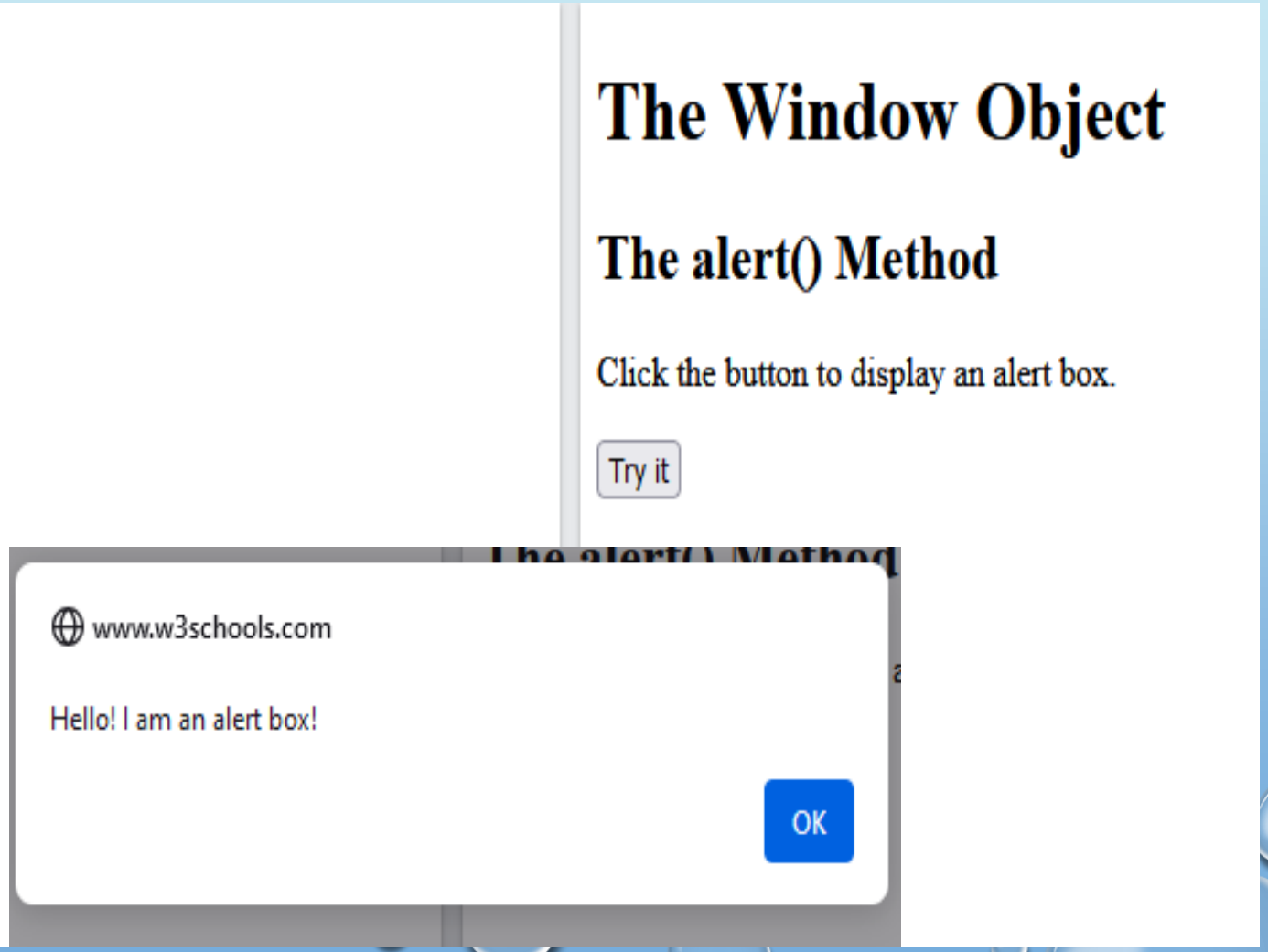
POPUP BOXES – ALERT

- AN ALERT BOX IS OFTEN USED IF YOU WANT TO MAKE SURE INFORMATION COMES THROUGH TO THE USER.
- WHEN AN ALERT BOX POPS UP, THE USER WILL HAVE TO CLICK "OK" TO PROCEED.

```
<script>
alert(“Good Morning!")
</script>
```

ALERT

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

<h1>The Window Object</h1>
<h2>The alert() Method</h2>

<p>Click the button to display an alert box.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  alert("Hello! I am an alert box!");
}
</script>

</body>
</html>
```



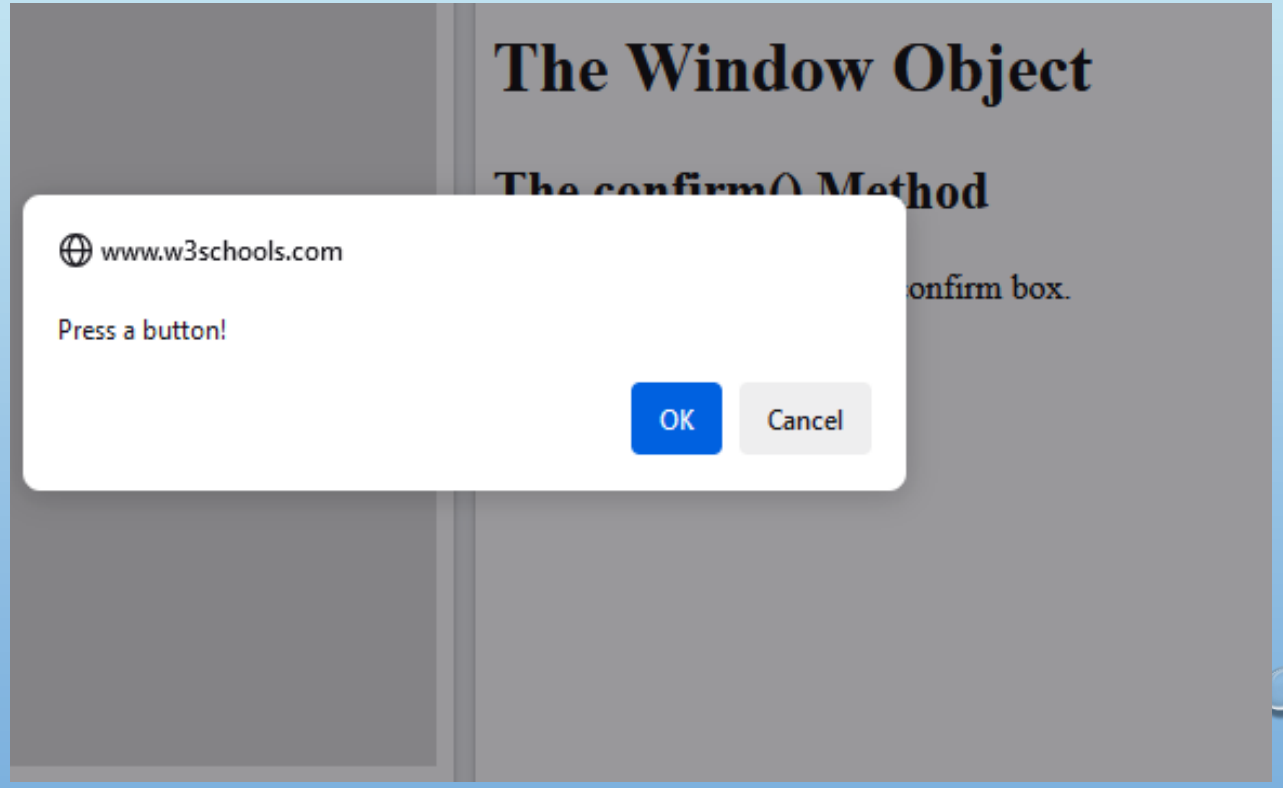
POPUP BOXES – CONFIRM

- A CONFIRM BOX IS OFTEN USED IF YOU WANT THE USER TO VERIFY OR ACCEPT SOMETHING.
- THE USER WILL HAVE TO CLICK EITHER "OK" OR "CANCEL“.
- <script>
  x=confirm (“Are you sure you want to delete ?”)
  </script>

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

<h1>The Window Object</h1>
<h2>The confirm() Method</h2>

<p>Click the button to display a confirm box.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  confirm("Press a button!");
}
</script>

</body>
</html>
```

CONFIRM


The Window Object
The confirm() Method
Click the button to display a confirm box.
Try it


The Window Object
www.w3schools.com
Press a button!
OK
Cancel

Java Script – Operators

Operator is used to apply some process over operands. E.g., 4+5, here 4 and 5 are operands, "+" is the operator. Types of operators are:

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

Java Script – Arithmetic Operators

Operator	Purpose	Example
+ (Addition)	Adds two operands	A + B => 8
- (Subtraction)	Subtracts the second operand from the first	A – B => 2
* (Multiplication)	Multiply both operands	A * B => 15
/ (Division)	Divide the numerator by the denominator	A / B => 1
% (Modulus)	Outputs the remainder of an integer division	A % B => 2
++ (Increment)	Increases an integer value by one	A++ => 6
-- (Decrement)	Decreases an integer value by one	A – – => 4

Here A has 5 and B has 3

Java Script – Relational Operators

Operator	Purpose	Example
== (Equals)	Checks if the value of two operands are equal or not, if yes, then the condition becomes true.	A == B => False
!= (Not Equals)	Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.	A != B => True
> (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.	A > B => True
< (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.	A < B => False
>= (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	A >= B => True
<= (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.	A <= B => False

Here A has 5 and B has 3

Java Script – Logical Operators

Operator	Purpose	Example
&& (and)	If both the operands are non-zero, then the condition becomes true	A && B => True (A > B) &&(A>C) => True (C >A) && (C>B) => False
|| (or)	If any of the two operands are non-zero, then the condition becomes true	A || B => True (A > B) || (A>C) => True (C >A) || (C>B) => True
! (Logical NOT)	Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.	!A => False !(A > B) => False !(C > A) => True

Here A has 5 , B has 3 and C has 4

Java Script – Assignment Operators

Operator	Purpose	Example
= (Simple Assignment)	Assigns values from the right side operand to the left side operand	C = A + B
+=(Add and Assignment)	It adds the right operand to the left operand and assigns the result to the left operand.	A = A+ B => A += B
–= (Subtract and Assignment)	It subtracts the right operand from the left operand and assigns the result to the left operand.	A = A - B => A -= B
*= (Multiply and Assignment)	It multiplies the right operand with the left operand and assigns the result to the left operand.	A = A* B => A *= B
/= (Divide and Assignment)	It divides the left operand with the right operand and assigns the result to the left operand.	A = A/ B => A /= B
%= (Modules and Assignment)	It takes modulus using two operands and assigns the result to the left operand.	A = A% B => A %= B

Java Script – Miscellaneous Operators

Operator	Purpose	Example
Conditional Operator (? :)	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.	Max =(A>B) ? A:B
typeof()	The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.	var A=20; var name="java"; typeof(A) => number typeof(name) => string

```
// Type conversion is performed before comparison
var v1 = ("5" == 5); // true
// No implicit type conversion.
// True if only if both types and values are equal
var v2 = ("5" === 5);// false

var v3 = (5 === 5.0); // true  var v4 = (true == 1); // true (true is
converted to 1)

var v5 = (true == 2); // false (true is converted to 1)

var v6 = (true == "1") // true
```

Java Script – Conditional & Branching

conditional statements allow the program to make correct decisions and perform right actions when it is required.

JavaScript supports the following forms of **conditional and branching** statement:

- **if statement**
- **if...else statement**
- **if...else if... statement**
- **switch statement**

Java Script – if … else statement

The if statement can be used to execute a block of JavaScript code for the given condition is true. if statement has additional part "else"; so that, it can execute another block of code for the given condition is false. "else" is an optional part.

Syntax:

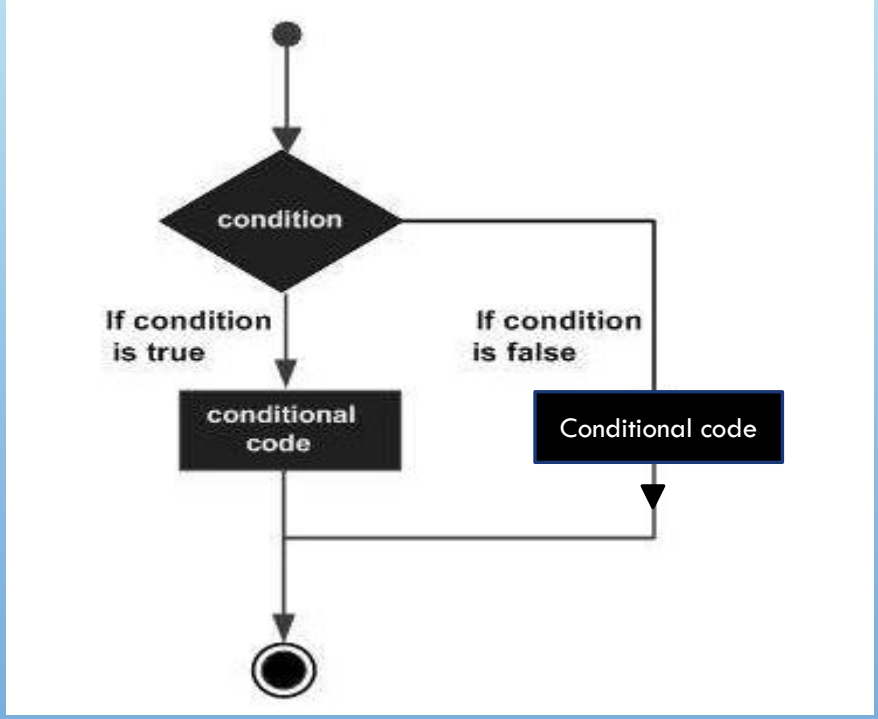
```
if (condition) {
    block of code to be executed if the condition is true
}
else
{
block of code to be executed if the condition is false
}
```

Example 1:

```
var age=12;
if (age < 18)
 category="Minor";
```

Example 2:

```
var a=10;
var b=5;
var max;
if(a > b)
  max = a;
else
  max = b;
```



Java Script – if…else nested

The **if…else if…** statement is an advanced form of **if…else** that allows JavaScript to make a correct decision out of several conditions.

```
Syntax:
if (expression 1) {
Statement(s) to be executed if expression 1 is true }
else if (expression 2) {
 Statement(s) to be executed if expression 2 is true }
else if (expression 3) {
Statement(s) to be executed if expression 3 is true }
else {
Statement(s) to be executed if no expression is true }
```

It is just a series of **if** statements, where each **if** is a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed

Java Script – if…else nested

Example:

```
var size = "S";
if( size == "S" ) { document.write("Size is Small"); }
else if( size == "M" ) { document.write("Size is Medium"); }
else if( size == "L" ) { document.write("Size is Large"); }
else { document.write("Size is above Large"); }
```

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript if .. else</h2>
<p>A time-based greeting:</p>
<script>
const time = new Date().getHours();
let greeting;
if (time < 10) {
  greeting = "Good morning";
}else if (time < 20) {
  greeting = "Good day";
}else {
  greeting = "Good evening";
}
document.write("<h1>" +greeting + "</h1>");
</script>
</body>
</html>
```

JavaScript if .. else

A time-based greeting:

Good day

Java Script – Switch …. Case Statement

The objective of a **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression.

The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

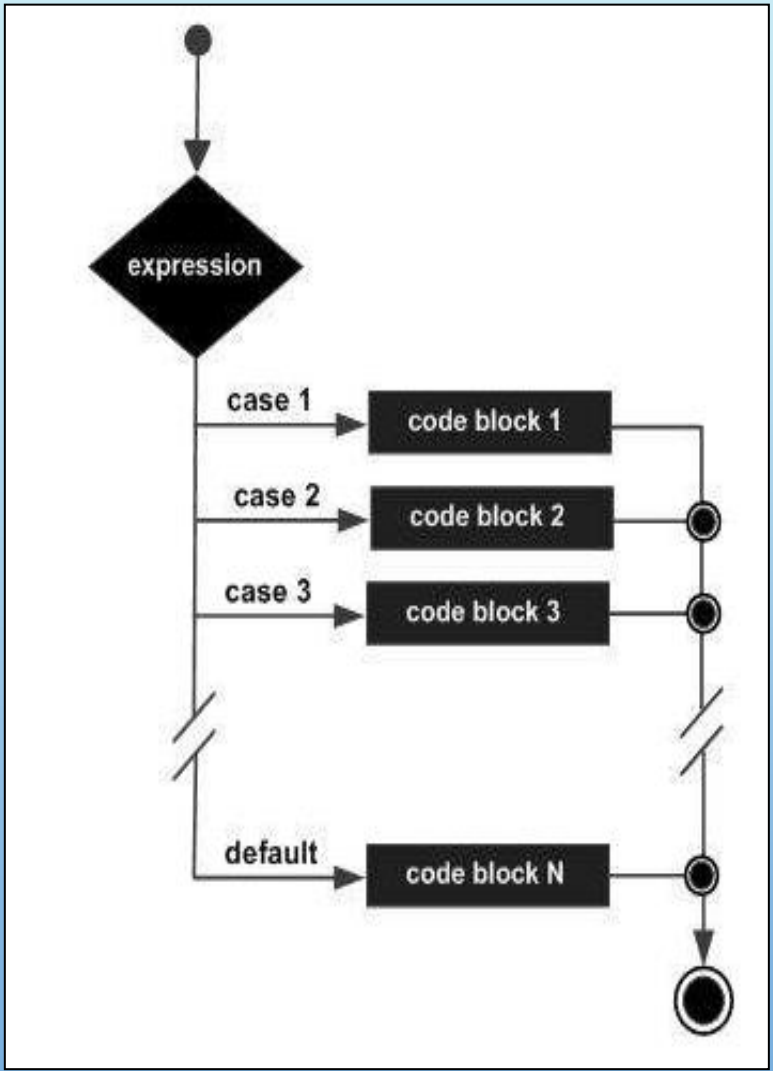
The **break** statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

```
switch (expression) {
   case condition 1: statement(s)
   break;

   case condition 2: statement(s)
   break;
   ...

   case condition n: statement(s)
   break;

   default: statement(s)
}
```

Java Script – Switch … Case Statement

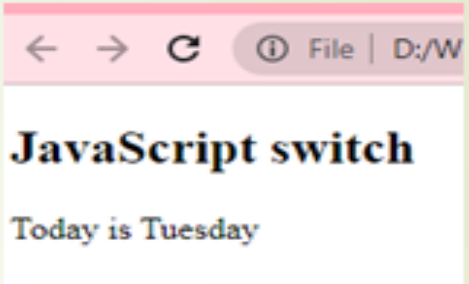
Example:

```
var day="wed";
switch (day) {
case "mon": document.write("Day is Monday");
            break;
case "tue": document.write("Day is Tuesday");
            break;
case "wed": document.write("Day is Wednesday");
            break;
case "thu": document.write("Day is Thursday");
            break;
case "fri": document.write("Day is Friday");
            break;
case "sat": document.write("Day is Saturday");
            break;
case "sun": document.write("Day is Sunday");
            break;
default: document.write("Unknown day"); }
```



SWITCH CASE DEMO

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript switch</h2>
<script>
let day;
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";
}
document.write("Today  is " + day);
</script>
</body>
</html>
```



new Date() returns the date based on the input parameter and Date() returns todays date on the browser.

Java Script – Conditional and Looping Statement

Conditional and looping is similar to branching statement but the difference is the block has repeated till the condition is true. This has classified into three categories:

- while loop
- do…while loop
- for loop

- Loops can execute a block of code a number of times.
- JavaScript supports different kinds of loops:

for - loops through a block of code a number of times

for/in - loops through the properties of an object

for/of - loops through the values of an iterable object

while - loops through a block of code while a specified condition is true

do/while - also loops through a block of code while a specified condition is true

Java Script – While … Loop

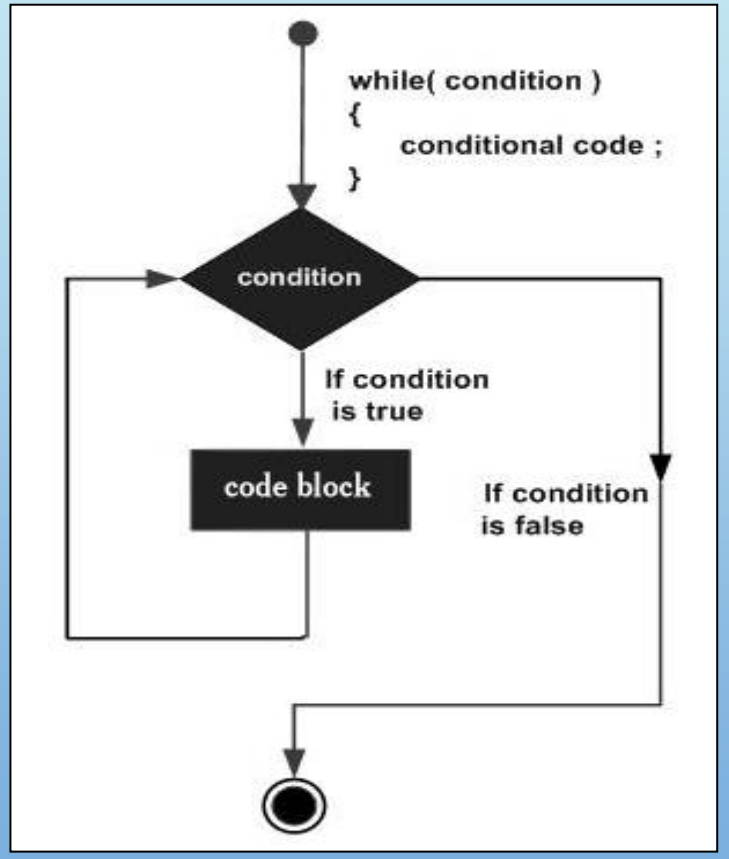
The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false,** the loop terminates.

Syntax:

```
while (expression) {
Statement(s) to be executed if
expression is true }
```

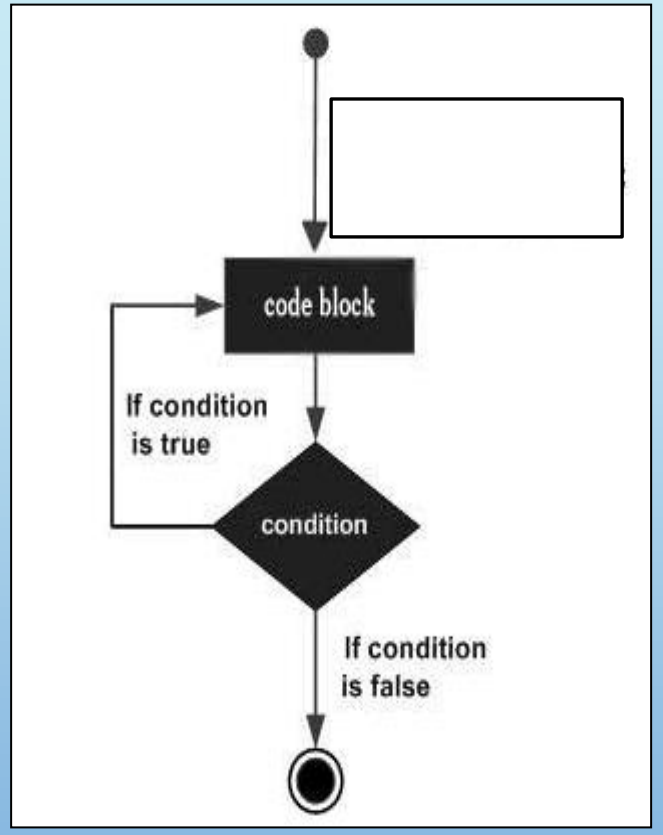
Example:

```
var count = 0; document.write("Starting Loop ");
while (count < 10) { document.write("Current
Count : " + count + "<br>");
count++;
 }
```



Java Script – do..while loop

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is **false**.



Syntax:

```
do {
Statement(s) to be executed;
} while (expression);
```

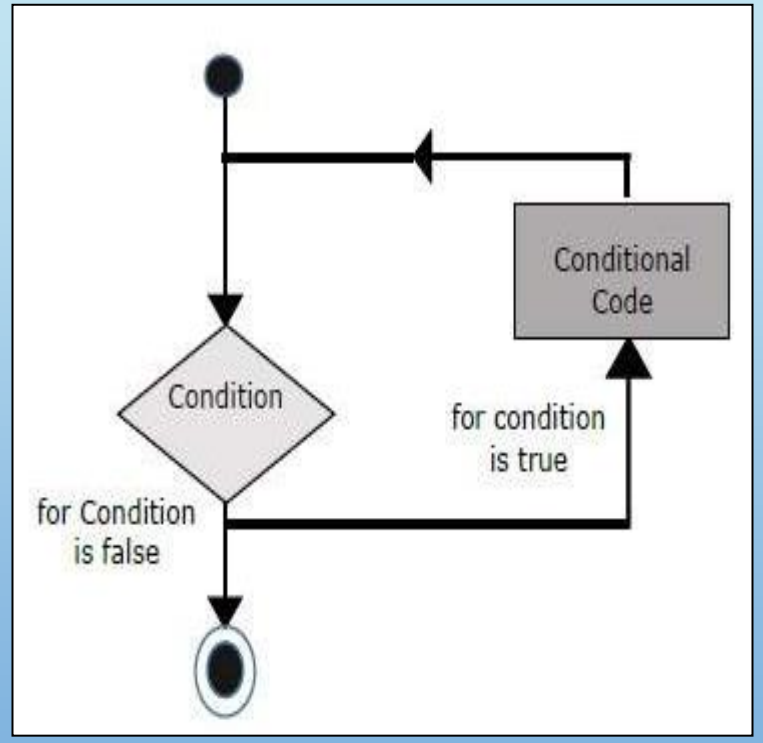
Example:

```
var count = 0;
document.write("Starting Loop" + "<br />");
do {
document.write("Current Count : " + count +
"<br />");
count++; }
while(count <5);
```

Java Script – for loop

The **'for'** loop is the most compact form of looping. It includes the following three important parts –

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.



Java Script – for loop

Syntax:

```
for (initialization; test condition; iteration statement)
{ Statement(s) to be executed if test condition is true }
```

Example:

```
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++) {
document.write("Current Count : " + count );
document.write("<br />");
}
```

FOR IN LOOP

- THE JAVASCRIPT FOR IN STATEMENT LOOPS THROUGH THE PROPERTIES OF AN OBJECT

- **FOR (KEY IN OBJECT) {**
- ***// CODE BLOCK TO BE EXECUTED***
- **}**

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript For In Loop</h2>
<p>The for in statement loops through the properties of an object:</p>
<script>
const person ={fname:"John", lname:"Doe", age:25};
let txt = "";
for (let x in person) {
  txt += person[x] + " ";
}
document.write("<h1>" +txt+ "</h1>");
</script>
</body>
</html>
```

JavaScript For In Loop

The for in statement loops through the properties of an object:

John Doe 25

FOR OF LOOP

- THE JAVASCRIPT FOR OF STATEMENT LOOPS THROUGH THE VALUES OF AN ITERABLE OBJECT.
- IT LETS YOU LOOP OVER ITERABLE DATA STRUCTURES SUCH AS ARRAYS, STRINGS, ETC
- **FOR (VARIABLE OF ITERABLE) {**
- ***// CODE BLOCK TO BE EXECUTED***
- **}**

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript For Of Loop</h2>
<p>The for of statement  loops through the values of any iterable object:</p>
<script>
const cars = ["BMW", "Volvo", "Mini"];
let text = "";
for (let x of cars) {
  text += x + "<br>";
}
document.write(text);
</script>
</body>
</html>
```

JavaScript For Of Loop

The for of statement loops through the values of any iterable object:

BMW
Volvo
Mini

DIFFERENCE BETWEEN FOR...OF AND FOR...IN

- THE MAIN DIFFERENCE BETWEEN THEM IS IN WHAT THEY ITERATE OVER.
- **THE FOR...IN STATEMENT ITERATES OVER THE ENUMERABLE STRING PROPERTIES OF AN OBJECT,**
- **WHILE THE FOR...OF STATEMENT ITERATES OVER VALUES THAT THE ITERABLE OBJECT DEFINES TO BE ITERATED OVER.**

CONTROL STATEMENTS -EXAMPLES

```
CONST NUM = 5;


IF (NUM > 0) {
    CONSOLE.LOG("THE NUMBER IS POSITIVE.");
};
```

```
LET NUM = -10;


IF (NUM > 0)

    CONSOLE.LOG("THE NUMBER IS POSITIVE.");

ELSE

    CONSOLE.LOG("THE NUMBER IS NEGATIVE");
```

```
// PROGRAM TO CHECK IF THE NUMBER IS POSITIVE

CONST NUMBER = PROMPT("ENTER A NUMBER: ");

// CHECK IF NUMBER IS GREATER THAN 0
IF (NUMBER > 0) {
    // THE BODY OF THE IF STATEMENT
    CONSOLE.LOG("POSITIVE NUMBER");
}

CONSOLE.LOG("NICE NUMBER");
```

```
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.");
};
```

```
// PROGRAM TO PRINT THE VALUE OF I

FOR (LET I = 1; I <= 5; I++) {

    // BREAK CONDITION
    IF (I == 3) {
        BREAK;
    }

    CONSOLE.LOG(I);
}
```

```
LET SUM = 0;
// INFINITE LOOP
WHILE (TRUE) {
    // GET NUMBER INPUT
    LET NUM = NUMBER(PROMPT("ENTER A NUMBER: "));

    // TERMINATE THE LOOP IF NUM IS NEGATIVE
    IF (NUM < 0)
        BREAK;
    }
    // OTHERWISE, ADD NUM TO SUM
    ELSE {
        SUM += NUM;
    }
}
// PRINT THE SUM
CONSOLE.LOG(`SUM: ${SUM}`);
```

JAVASCRIPT EXPLICIT CONVERSION

```
LET RESULT;

// CONVERT STRING TO NUMBER
RESULT = NUMBER("5");
CONSOLE.LOG(RESULT, "-", TYPEOF(RESULT));

// CONVERT BOOLEAN TO STRING
RESULT = STRING(TRUE);
CONSOLE.LOG(RESULT, "-", TYPEOF(RESULT));

// CONVERT NUMBER TO BOOLEAN
RESULT = BOOLEAN(0);
CONSOLE.LOG(RESULT, "-", TYPEOF(RESULT));
```

Java Script – Functions

- A function is a group of reusable code which can be called anywhere in the program. This eliminates the need of writing the same code again and again.
- **Definition** – The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.
- **Calling** – To invoke a function somewhere later in the script, it should be called by writing the name of that function with () and arguments if any.
- **Return** – A JavaScript function can have an optional return statement. This is required when return a value from the function. This statement should be the last statement in a function.

Syntax:

```
function
functionname(parameter-list)
{
 statements
}
```

```
<script type="text/javascript">
document.write(Date());
</script>
```

Mon Sep 02 2024 10:47:07 GMT+0530 (India Standard Time)

Java Script – Functions

Parameters

```
function sayHello(name, age) {
document.write (" name: " + name);
document.write("Age is: " + age);
}
...........
...........
...........

<input type = "button" onclick =
"sayHello('Zara', 7)" value = "Say Hello">
```

Function Definition

Arguments

Function Calling

Java Script – Functions

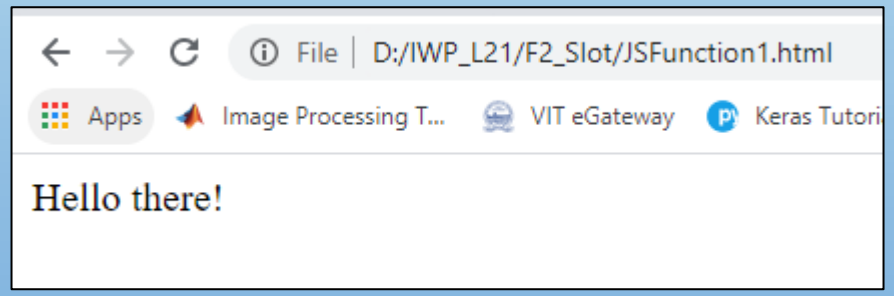
Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

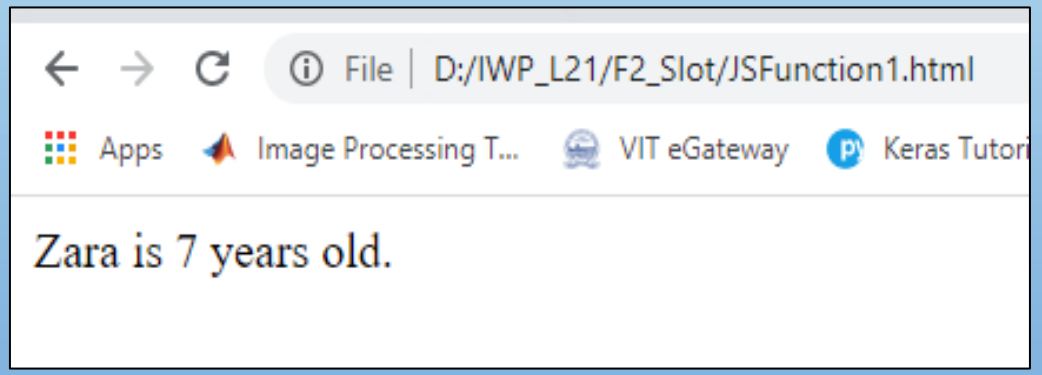
Java Script – Simple Function Example

```
<html>
<script type = "text/javascript">
function sayHello() {
 document.write ("Hello there!");
}
</script>
<body>
<p>Click the following button to call the
function
</p>
<form>
 <input type = "button" onclick = "sayHello()"
value = "Say Hello">
</form> </body>
</html>
```



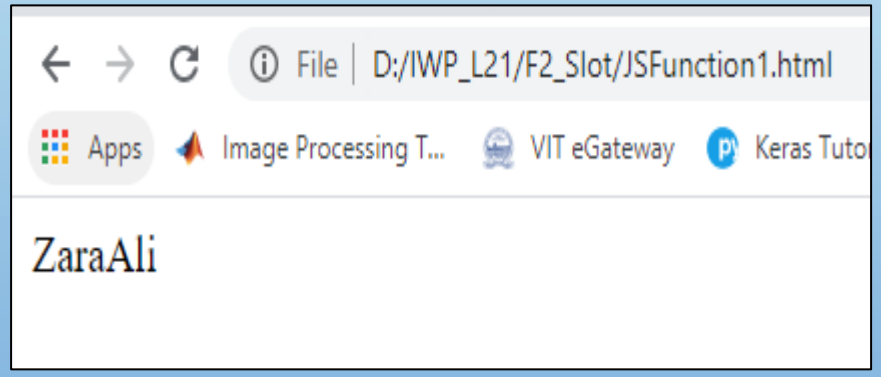
Java Script – Function with Parameters

```
<html> <head>
<script type = "text/javascript">
function sayHello(name, age) {
document.write (name + " is " + age + "
years old.");
}
</script>
</head>
<body>
 <p>Click the following button to call the
function</p>
<form>
<input type = "button" onclick =
"sayHello('Zara', 7)" value = "Say Hello">
</form>

 </body> </html>
```



Java Script – Function with return

```
<html> <head>
<script type = "text/javascript">
 function concatenate(first, last) {
var full;
full = first + last;
return full; }

function secondFunction() {
var result;
result = concatenate('Zara', 'Ali');
document.write (result );
}
 </script>
 </head> <body>
<p>Click the following button to call the
function</p>
<form>
<input type = "button" onclick =
"secondFunction()" value = "Call Function">
</form>
</body> </html>
```



Java Script – Arrays

- An array is a special variable, which can hold more than one value at a time. It stores a fixed-size sequential collection of elements.

  var *array_name* = [*item1*, *item2*, ...];

- Using an array literal is the easiest way to create a JavaScript Array.

  var cars = ["Saab", "Volvo", "BMW"];

- Access an array element by referring to the **index number**.

  var name = cars[0];

  Cars[0] = "Ambassdor"

- Arrays are treated as objects; Objects use **names** to access its "members".

  var array_object = {key1:value1, key 2:value2,… key n:valuen}

  var person = {firstName:"John", lastName:"Doe", age:46};

Java Script – Array Properties

- The length property of an array returns the length of an array

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var arraySize = fruits.length;
document.write("The array size is "+arraySize);
```

Java Script – Array Methods

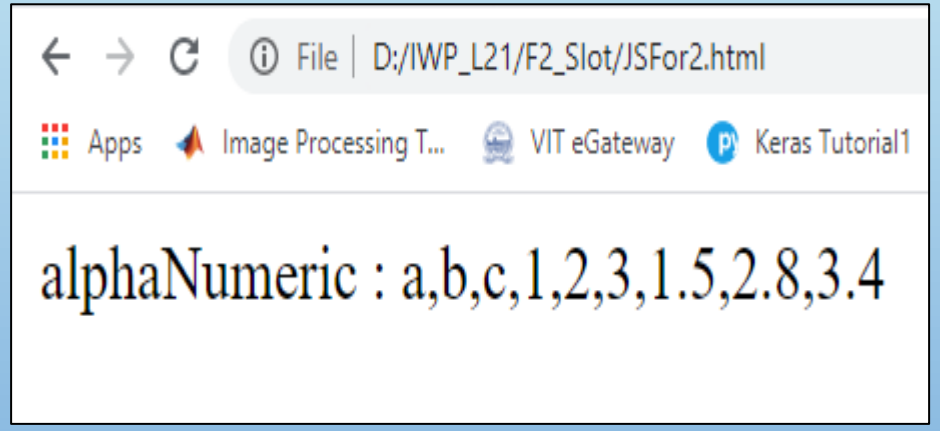
Method	Description
arrayX = Array1.concat(array2 [, array3,..])	Returns a new array comprised of this array joined with other array(s) and/or value(s).
intVar = Array.indexOf(value)	Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
Str1 = Array.join()	Joins all elements of an array into a string.
intVar = lastIndexOf(value)	Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
Var1 = array.pop()	Removes the last element from an array and returns that element.
Array.push(val1[,val2, ..])	Adds one or more elements to the end of an array and returns the new length of the array.

Java Script – Array Methods

Method	Description
newArray = array.reverse()	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
Var1 = array.shift()	Removes the first element from an array and returns that element.
newArray = array.slice(start, end)	Extracts a section of an array and returns a new array.
Array.sort()	Sorts the elements of an array
Array.splice(index, howmany, newitem1,newitem2,…)	Adds and/or removes elements from an array.
stringvar = array.toString()	Returns a string representing the array and its elements.
Array.unshift(val1[,val2, …])	Adds one or more elements to the front of an array and returns the new length of the array.

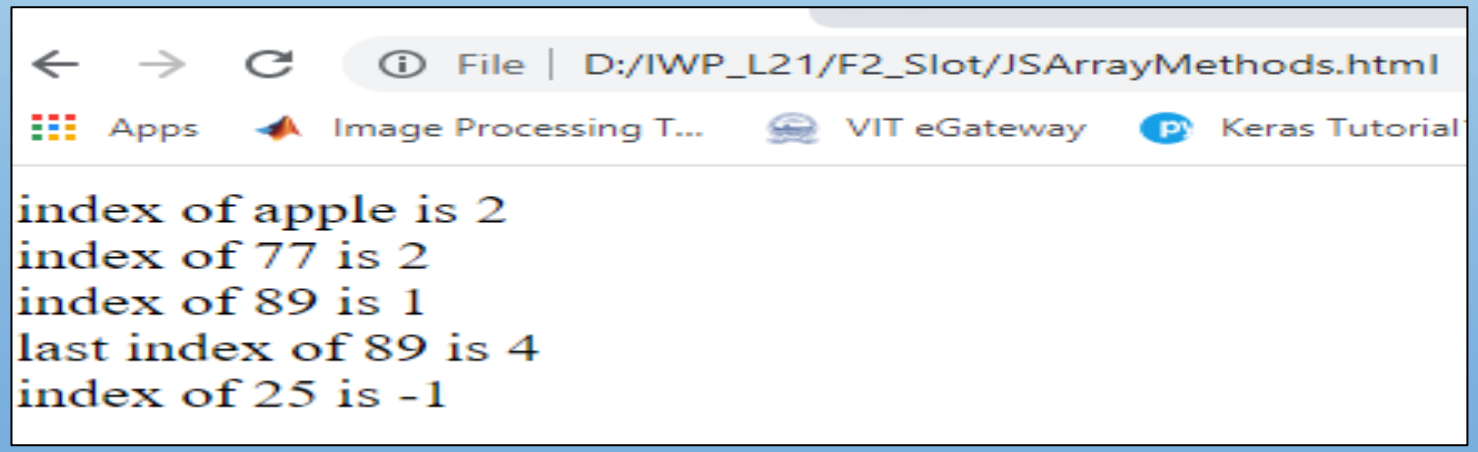
Java Script – concat()

```
<html>
<script type = "text/javascript">
var alpha = ["a", "b", "c"];
var numeric = [1, 2, 3];
var floatvalues= [1.5, 2.8, 3.4];
var alphaNumeric = alpha.concat(numeric,
floatValues);
document.write("alphaNumeric : " + alphaNumeric );
</script>
</html>
```



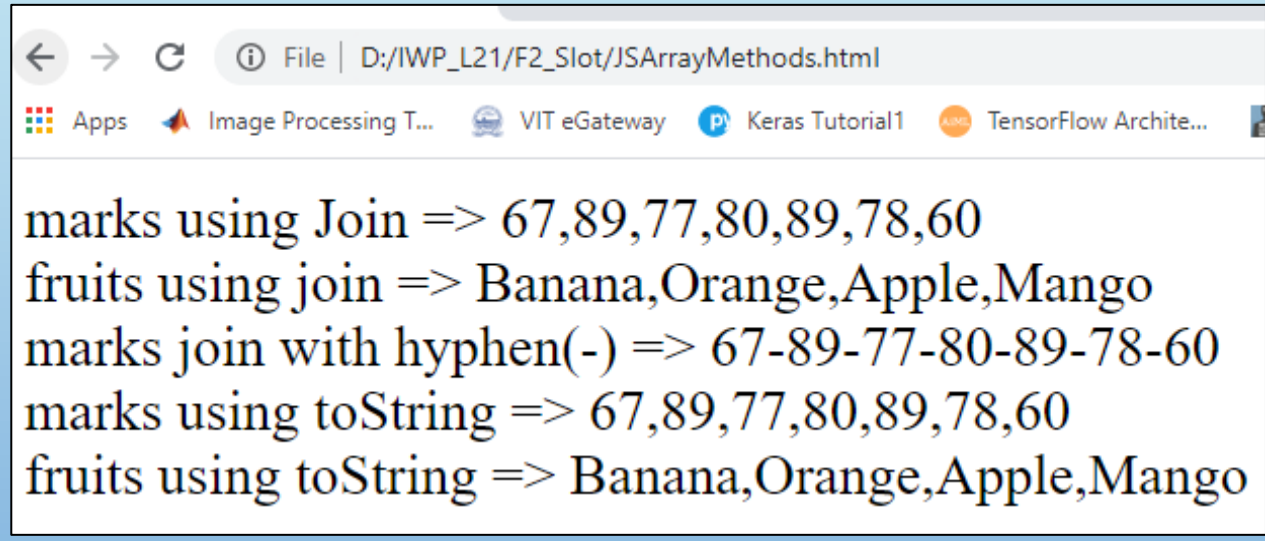
Java Script – indexOf & lastIndexOf

```
<script>
  var fruits = ["Banana", "Orange", "Apple", "Mango"];
  var marks =[67,89,77,80,89,78,60];
  var a = fruits.indexOf("Apple");
  document.write("index of apple is "+a+"<br>");
  document.write("index of 77 is "+ marks.indexOf(77)+"<br>");
  document.write("index of 89 is "+ marks.indexOf(89)+"<br>");
  document.write("last index of 89 is "+ marks.lastIndexOf(89)+"<br>");
</script>
```



Java Script – join and toString

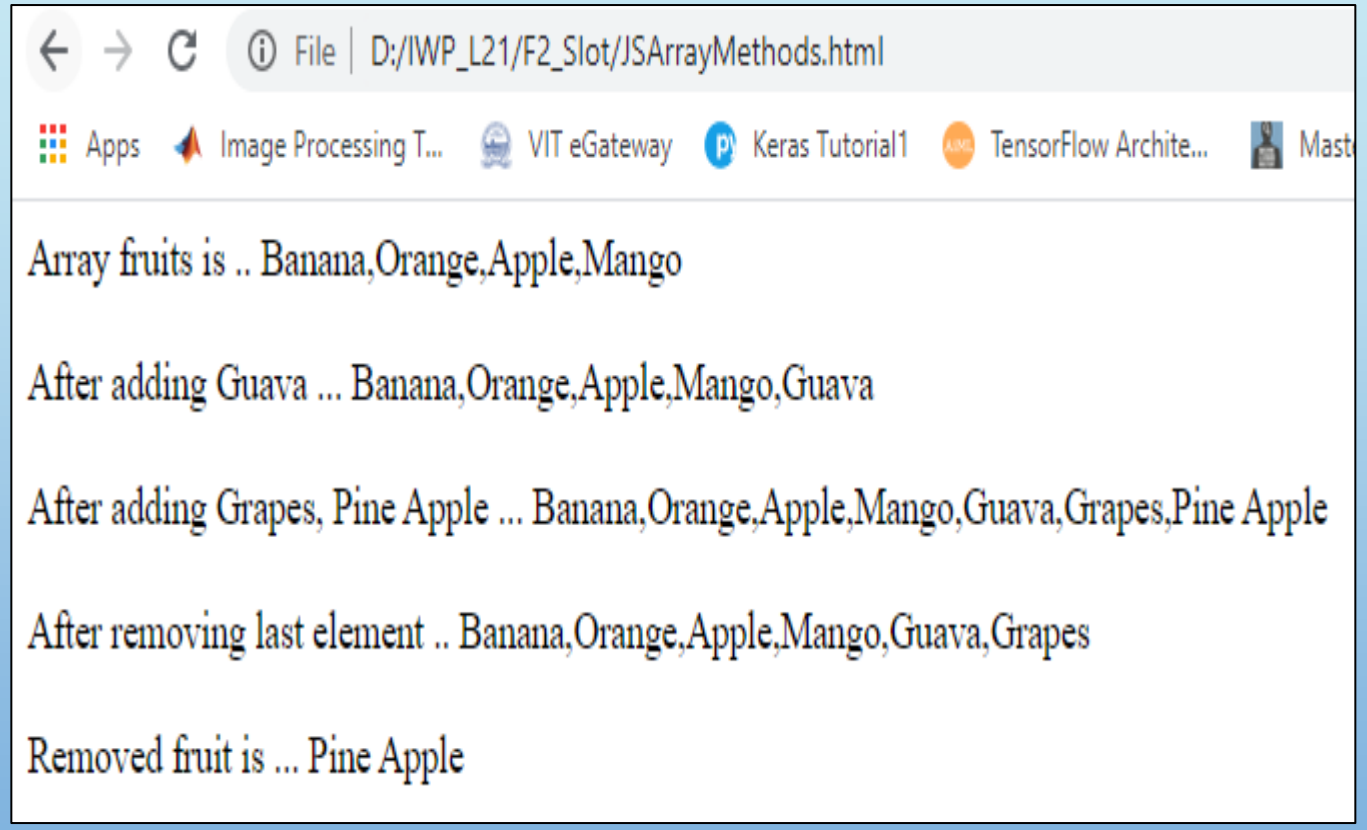
```
<script>
  var fruits = ["Banana", "Orange", "Apple", "Mango"];
  var marks =[67,89,77,80,89,78,60];
  var marksJoin = marks.join();
  var fruitsJoin = fruits.join();
  var marksJoinHyphen = marks.join("-");
  var marksToString = marks.toString();
  var fruitsToString = fruits.toString();
document.write("marks using Join => "+marksJoin+"<br>");
document.write("fruits using join => "+fruitsJoin+"<br>");
document.write("marks join with hyphen(-) =>
"+marksJoinHyphen+"<br>");
document.write("marks using toString =>
"+marksToString+"<br>");
document.write("fruits using toString =>
"+fruitsToString+"<br>");
</script>
```



The toString() method returns a string as a string.
The toString() method does not change the original string.

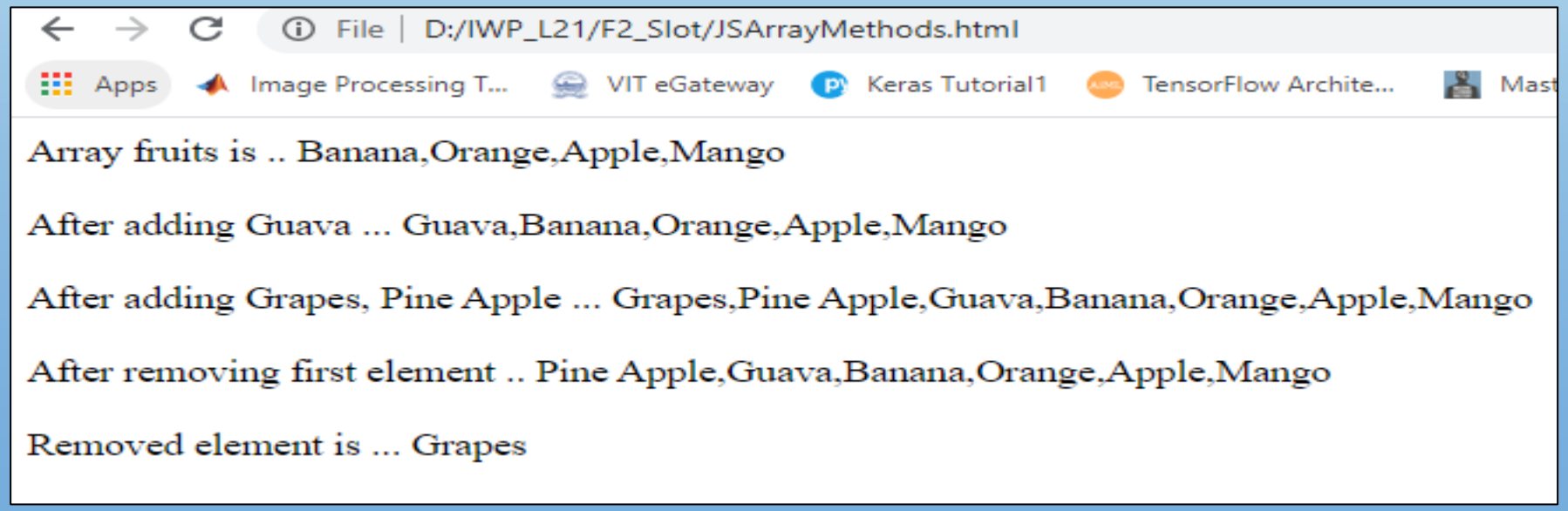
Java Script – push & pop

```
<script>
  var fruits = ["Banana", "Orange", "Apple",
"Mango"];
 document.write("Array fruits is ..
"+fruits+"<br><br>");
 fruits.push("Guava");
 document.write("After adding Guava ...
"+fruits+"<br><br>");
 fruits.push("Grapes","Pine Apple");
document.write("After adding Grapes, Pine Apple ...
"+fruits+"<br><br>");
var popFruit = fruits.pop();
document.write("After removing last element ..
"+fruits+"<br><br>");
document.write("Removed fruit is ... "+
popFruit+"<b>");
</script>
```



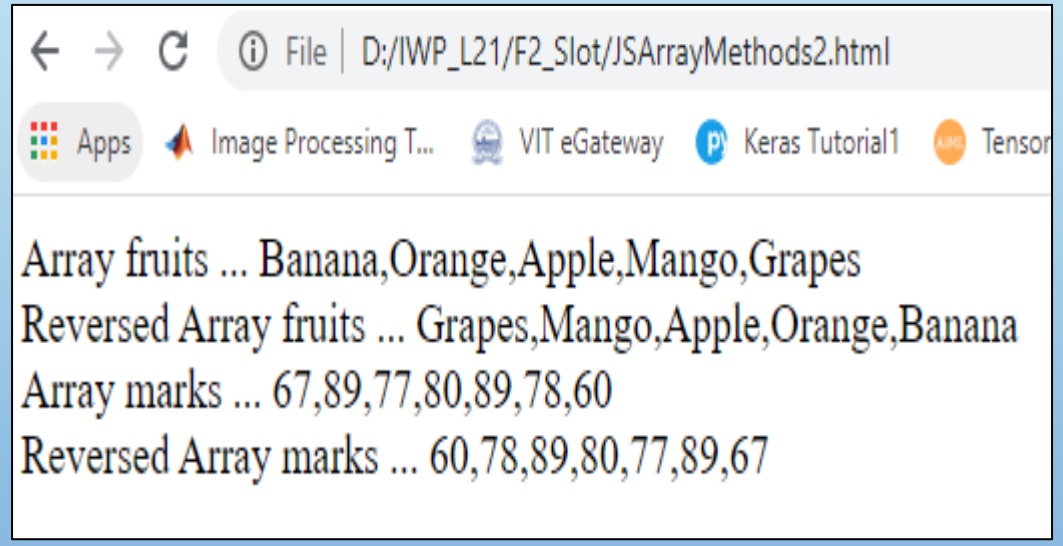
Java Script – unshift & shift

```
<script>
  var fruits = ["Banana", "Orange", "Apple", "Mango"];
 document.write("Array fruits is .. "+fruits+"<br><br>");
 fruits.unshift("Guava");
 document.write("After adding Guava ... "+fruits+"<br><br>");
 fruits.unshift("Grapes","Pine Apple");
document.write("After adding Grapes, Pine Apple ... "+fruits+"<br><br>");
var popFruit = fruits.shift();
document.write("After removing first element .. "+fruits+"<br><br>");
document.write("Removed element is ... "+ popFruit+"<b>"); </script>
```



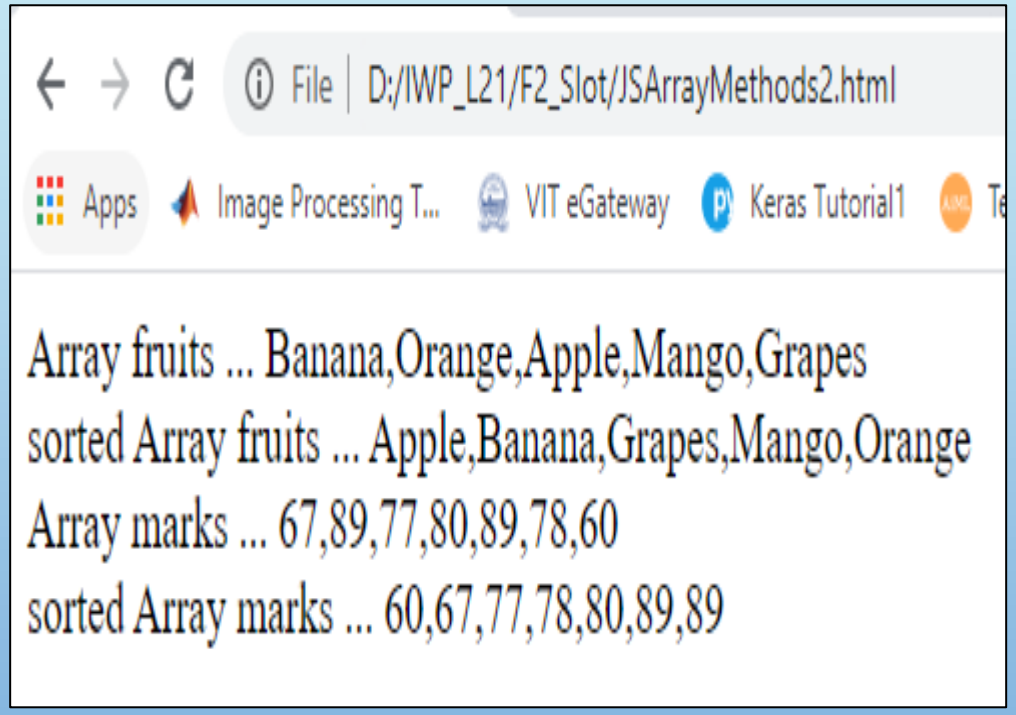
Java Script – reverse

```
<script>
  var fruits = ["Banana", "Orange", "Apple", "Mango","Grapes"];
 var marks =[67,89,77,80,89,78,60];
  document.write("Array fruits ... "+ fruits+"<br>");
 fruits.reverse();
 document.write("Reversed Array fruits ... "+ fruits+"<br>");
 document.write("Array marks ... "+ marks+"<br>");
 marks.reverse();
 document.write("Reversed Array marks ... "+ marks+"<br>");
</script>
```



Java Script – sort

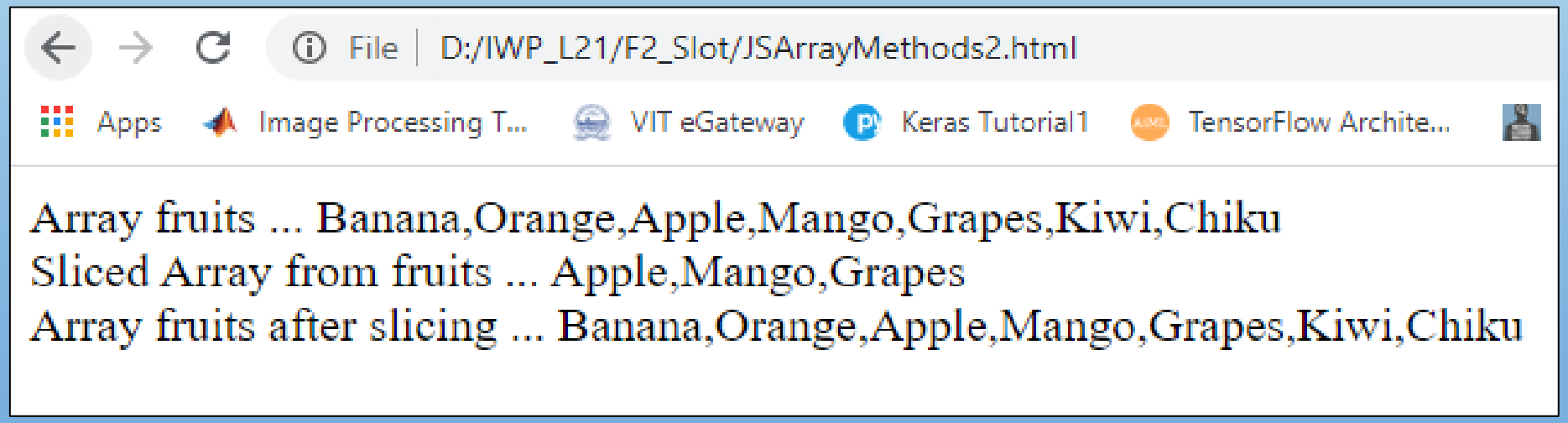
```
<script>
 var fruits = ["Banana", "Orange", "Apple", "Mango","Grapes"];
 var marks =[67,89,77,80,89,78,60];
 document.write("Array fruits ... "+ fruits+"<br>");
 fruits.sort();
 document.write("sorted Array fruits ... "+ fruits+"<br>");
 document.write("Array marks ... "+ marks+"<br>");
 marks.sort();
 document.write("sorted Array marks ... "+ marks+"<br>");
</script>
```



Java Script – slice

```
<script>
 var fruits = ["Banana", "Orange", "Apple",
"Mango","Grapes","Kiwi","Chiku"];
 document.write("Array fruits ... "+ fruits+"<br>");
 var slicedFruits = fruits.slice(2,5);
 document.write("Sliced Array from fruits ... "+
slicedFruits+"<br>");
 document.write("Array fruits after slicing ... "+ fruits+"<br>");
</script>
```

The slice() method returns selected elements in an array, as a new array.
The slice() method selects from a given *start*, up to a (not inclusive) given *end*.
The slice() method does not change the original array.



Java Script – splice

```
<script>
 var fruits = ["Banana", "Orange", "Kiwi","Chiku"];
 document.write("Array fruits ... "+ fruits+"<br>");
 fruits.splice(2,0,"Apple", "Mango","Grapes");
 document.write("fruits Spliced with \"Apple\",
\"Mango\",\"Grapes\" ... "+ fruits+"<br>");
 fruits.splice(3,1);
 document.write("Array fruits after empty splicing at 3... "+
fruits+"<br>");
</script>
```

String object

- The String object is used to manipulate a stored piece of text.

```
var txt="Hello world!  world!";
document.write(txt.length);  //12
document.write(txt.toUpperCase()); //HELLO WORLD!
document.write(txt.match("world") ); //world
document.write(txt.match("World") ) ;//null
document.write(txt.indexOf("world")); //6
var str="Visit Microsoft!";
document.write(str.replace("Microsoft","CTS")); //Visit CTS!
```

STRING OBJECT & METHODS

- METHODS
  - REPLACE(EXP [, REPLACEMENT TEXT]);

```
name = “Hello World”;
message=name.replace(/World/,”VIT”);        // Hello VIT
```

  - SEARCH(EXPRESSION)
  - SPLIT(DELIMITER[,COUNT]); // AN ARRAY OF SUBSTRING

```
name = “Hello VIT”;
message=name.split(“ ”);        // Hello, VIT
```

  - SUBSTR(START[,LENGTH]) ;
  - TOUPPERCASE()
  - TOLOWERCASE();

```
name = “Hello VIT”;
message=name.toLowerCase();
```

ARRAY OBJECT & METHODS

Methods	Description
concat()	Joins two or more arrays, and returns a copy of the joined arrays
indexOf()	Search the array for an element and returns its position
join()	Joins all elements of an array into a string
lastIndexOf()	Search the array for an element, starting at the end, and returns its position
pop()	Removes the last element of an array, and returns that element
push()	Adds new elements to the end of an array, and returns the new length
reverse()	Reverses the order of the elements in an array
shift()	Removes the first element of an array, and returns that element
slice()	Selects a part of an array, and returns the new array
sort()	Sorts the elements of an array
splice()	Adds/Removes elements from an array
toString()	Converts an array to a string, and returns the result
unshift()	Adds new elements to the beginning of an array, and returns the new length
valueOf()	Returns the primitive value of an array

MATH OBJECT & METHODS

Methods	Description
abs(x)	Returns the absolute value of x
ceil(x)	Returns x, rounded upwards to the nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x
floor(x)	Returns x, rounded downwards to the nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x,y,z,...,n)	Returns the number with the highest value
min(x,y,z,...,n)	Returns the number with the lowest value
pow(x,y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds x to the nearest integer
sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

DATE OBJECT

- THE DATE OBJECT IS USED TO WORK WITH DATES AND TIMES.
- DATE OBJECTS ARE CREATED WITH THE DATE() CONSTRUCTOR.

```
new Date() // current date and time
new Date(milliseconds) //milliseconds since 1970/01/01
new Date(yy,mm,dd) //with specified date
new Date(yy,mm,dd,hh,mm,ss) //with specified date & time
new Date("Month dd,yyyy") //datestring in the given format
new Date("Month dd,yyyy hh:mm:ss") //datestring in the
    //given format

var today = new Date()
var d1 = new Date("October 13, 1975 11:13:00")
var d2 = new Date(13,5,24)
var d3 = new Date(13,5,24,11,33,0)
```

DATE OBJECT-METHODS

- GETFULLYEAR() //4-DIGIT YEAR
- GETMONTH() //(0-11 AS JAN=0,FEB=1…)
- GETDATE() //(1-31)
- GETDAY() //(0-6 AS SUNDAY=0)
- GETHOURS //(0-23)
- GETMINUTES //(0-59)
- GETSECONDS //(0-59)
- GETMILLISECONDS //(0-999)
- GETTIMEZONEOFFSET //TIME DIFF BETWEEN LOCAL PC AND GMT

CURRENT DATE AND TIME

```
<script>
      var currentDate = new Date()
      var day = currentDate.getDate();
      var month = currentDate.getMonth() + 1;
      var year = currentDate.getFullYear();
      var my_date = day+"-"+month+"-"+year;
      document.write("Todays date is : "+my_date);

      var hours = currentDate.getHours()
      var minutes = currentDate.getMinutes()
      if (minutes < 10){
          minutes = "0" + minutes
      }
      document.write(hours + ":" + minutes + " ")
      if(hours > 11){
          document.write("PM")
      } else {
          document.write("AM")
      }
</script>
```

NUMBER OBJECT & METHODS

Property	Description
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
NaN	Represents a "Not-a-Number" value
POSITIVE_INFINITY	Represents infinity (returned on overflow)

NUMBER OBJECT & METHODS

Methods	Description
toExponential(x)	Converts a number into an exponential notation
toFixed(x)	Formats a number with x numbers of digits after the decimal point
toPrecision(x)	Formats a number to x length
toString()	Converts a Number object to a string

GLOBAL PROPERTIES AND METHODS

Global properties and functions can be used with all the built-in JavaScript objects.

Property	Description
Infinity	A numeric value that represents positive/negative infinity
NaN	"Not-a-Number" value
undefined	Indicates that a variable has not been assigned a value