ILAHIA COLLEGE OF ENGINEERING AND TECHNOLOGY

WE TECHNOLOGIES

Module IV

Introduction to JavaScript and jQuery :**The Basics of JavaScript:** Overview of JavaScript, Object Orientation and JavaScript, General Syntactic Characteristics- Primitives, Operations, and Expressions, Screen Output and Keyboard Input, Control Statements, Object Creation and Modification, Arrays, Functions. Callback Functions, Java Script HTML DOM.

Introduction to jQuery: Overview and Basics.

JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Mozilla, Firefox, Netscape, Opera.

4.1 Overview of JavaScript

ORIGIN

→Originally developed at Netscape by Brendan Eich, as LiveScript

→Initially named Mocha, later renamed as LiveScript

→ Became a joint venture of Netscape and Sun in 1995, renamed JavaScript

→Now standardized by the European Computer Manufacturers Association as ECMA-262 (also ISO 16262)

Javascript is divided into three parts

1) **Core Javascript** - Heart of the language , including its operators, expressions, statements and subprogram.

2) **Client-side JavaScript** - Collections of JavaScript code scripts , not programs, support the control of browsers and interaction with the user

3) **Server side Javascript** - Collection of objects that make the language useful on a web server.

2 Java and JavaScript

- Java and JavaScript are two completely different languages in both concept and design!
- Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.
- Object model of Javascript is quite different from java & C++.
- Java is strongly typed and Javascript is Dynamically typed.
- In Java ,types are all known at compile time, where as in Javascript variables need not be declared and compile time type checking impossible.
- Objects in Java are static where as Javascript objects are dynamic.

3. Uses of JAVASCRIPT

- JavaScript was designed to add interactivity to HTML
- JavaScript is a scripting language
- It is usually embedded directly into HTML pages
- JavaScript is an interpreted language
- Provide programming capability at both the server and the client ends of a Web connection.
- Improve appearance especially graphics
- Perform calculations
- Validation of input
- Less server interaction – we can validate the user i/p before sending the page off to the server. This saves server traffic.

4.Web Browsers & HTML- JAVASCRIPT Doc

- When a JavaScript code is encountered in the HTML document, the browser uses its JavaScript interpreter to "execute" the script. Output from the script becomes the next markup to be rendered. When the end of the script is reached, the browser goes back to reading the HTML document and displaying its content.
- There are two different ways to embed JavaScript in an HTML document:
  - **implicitly** and **explicitly.**
- In **explicit embedding**, the JavaScript code physically resides in the HTML document.
- In **implicit embedding** the JavaScript can be placed in its own file, separate from the HTML document.
- When JavaScript scripts are explicitly embedded, they can appear in either part of an HTML document—the head or the body
- The interpreter notes the existence of scripts that appear in the head of a document, but it does not interpret them while processing the head. This method is useful when using Javascript functions. The function can be defined in the head of the HTML document.Scripts that are found in the body of a document are interpreted as they are found.

4.2 Object Orientation and JavaScript

- JavaScript is NOT an object-oriented programming language, it is an object based language.
- JavaScript does not have classes

- Without classes Javscript does not support class-based inheritance
- Without class based inheritance it cannot support polymorphism
- It uses a technique that can be used to simulate some aspects of inheritance
- This is done with prototype object. This form of inheritance is called prototype-based inheritance.
- JavaScript has primitives for simple types
- The root object in JavaScript is Object – all objects are derived from Object
- All JavaScript objects are accessed through references

JavaScript Objects

- In JavaScript, objects are collections of properties, which correspond to the members of classes in Java and C++.
- Each property is either a data property(property) or a function or method.
- **JavaScript are embedded in HTML documents.**

Data properties appear in two categories:

primitive values and references to other objects.

- All objects in a JavaScript program are indirectly accessed through variables(objects).
- Such a variable is like a reference in Java. All primitive values in JavaScript are accessed directly
- For example, if myCar is a variable referencing an object that has the property engine, the engine property can be referenced with myCar.engine.
- The root object in JavaScript is Object. It is the ancestor, through prototype inheritance, of all objects. Object is having some methods but no data properties.
- A JavaScript object appears, as a list of **property–value pairs.**
- The properties are names; the values are data values or functions.
- All functions are objects and are referenced through variables.
- The collection of properties of a JavaScript object is dynamic: Properties can be added or deleted at any time.

How to Put a JavaScript Into an HTML Page?

```
<!DOCTYPE html>
<html lang="EN">
<head><title>My Javascript page</title></head>
<body>
<script type="text/javascript">
```

```
<!--
Javascript code
// -->
</script>
</body>
</html>
```

3.General Syntactic Characteristics

- JavaScript are embedded in HTML documents
- Either directly, as in

```
<script type = "text/javaScript">
        <! -- JavaScript script      //-- >
    </script>
```

- Or indirectly, as a file specified in the src attribute of <script>, as in

```
<script type = "text/javaScript"  src = "myScript.js">
</script>
```

Identifier form: begin with a letter or underscore, or dollar sign ($) followed by any number of letters, underscores, and digits

- Case sensitive
- 25 reserved words, plus future reserved words Eg:

  Break, continue, switch, case, if ,else, while, for…

- Comments: both // and /* … */
- Scripts are usually hidden from browsers that do not support JavaScript interpreters by putting them in special comments.

```
<!--
    -- JavaScript script –
//-->
```

- Semicolons can be a problem. They are "somewhat" optional.

Problem: When the end of the line may not be the end of a statement – JavaScript puts a semicolon there. But this implicit insertion can lead to problems

Eg: return

x;

The interpreter will insert a semicolon after return, because return need not be followed by an expression, making x an invalid orphan.

To avoid this problem, put each statement on its own line and terminate each statement with a semicolon.

When separated by semicolons, multiple statements on one line are allowed: a = 5; b = 6; c = a + b;

- **Javascript is case sensitive**
- **Test,TEST,Test all are distinct names**

//helloworld.html

```
<!DOCTYPE html>
<html lang="EN">
<head><title>My Javascript page</title>
<meta charset="utf-8" />
</head>
<body>
<script type="text/javascript">
        document.write("Hello World!");
</script>
</body>
</html>
```

4.4 Primitives, Operations, & Expressions

1.Primitive Types

Five primitive types: **Number, String, Boolean, Undefined**, or **Null.**

Javascript includes predefined objects that are closely related to primitive types named Number ,string and Boolean. These objects are called wrapper objects. Because each objects contains a property that stores a value of the corresponding primitive type. The purpose of Wrapper objects are used to provide properties and methods that are convenient to use with the values of primitive types. Javascript coerces values between the number type primitive values and number objects and between the string type primitive values and string

objects. The methods of Number and String objects can be used on variables of the corresponding primitive types.

The difference between primitives and objects

Suppose that prim is a primitive variable with the value 17 and obj is a Number object with property value 17. *prim* and *obj* are stored differently .

2. **Numeric & String Literals**

Numeric literals – like Java All numeric values are stored in double-precision floating point. Numeric values in Javascript are often called numbers. Numeric literals can be either integer or floating point values.

72,7.2,.72,7E2,7e2 etc are valid numeric literals.

String literals are delimited by either ' or " .Can include escape sequences (e.g., \n,\t).

Var str1=”Hello”;

Var str2=’Hello’;

To include a single quote character in a string delimited by single quotes, the embedded single quote must be preceded by a backslash.

```
'You \'re the best person I \'ve ever seen'
```

A double quote can be embedded in a double quoted string literal by preceding it with a backslash.

- A null string (a string with no characters) can be denoted with ‘ ‘ or “ “.
- All string literals are primitive types.

3. Other Primitive Types

Boolean values are true and false.**Used for evaluating Boolean Expressions.**

Var boo1= true;

Var boo2=false;

The only value of type Null is reserved word **null which** means no value. A variable is Null if it has not been explicitly declared or assigned a value.

The only value of type Undefined is **undefined.** If a variable has been explicitly declared, but not assigned a value it has the value undefined.

A variable declared without a value will have the value **undefined**.

ex) var carName;

Document.write(carName);

4.**Declaring variables**

JavaScript is dynamically typed – any variable can be used for anything ie variable are not typed but values are typed. A variable can have values of any primitive type, or it can be a reference to any objects .The interpreter determines the type of the value of a particular occurrence of a variable. Variables can be implicitly or explicitly declared.

- A variable can be declared either by assigning it a value**(implicit)**in which case the interpreter implicitly declares it to be a variable or by listing it in a declaration statement that begins with the reserved word **var(explicit)**

var sum =0,today ="Monday",flag = false;

var counter;

- Initial values can be included in a **var** declaration
- **Rules for variable names:**
  - **Variable names are case sensitive**
  - **They must begin with a letter or the underscore character**

strname – STRNAME (not same)

5.Numeric Operators

5.1Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2,5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

- Eg:increment operator

If the variable a has the value 7,

(++a)*3 =24

(a++)*3 = 21

In both cases a is set to 8

The *precedence rules* of a language specify which operator is evaluated first when two operators with different precedence are adjacent in an expression.

The *associativity rules* of a language specify which operator is evaluated first when two operators with the same precedence are adjacent in an expression.

- **Precedence and associativity**
- **Operator** **Associativity**

  ++,--,unary-,unary+ **Right(though it is irrelevant)**

  ***,/,%** **Left**

  Binary +,Binary - **Left**

  >,<,>=,<= **Left**

  ==,!= **Left**

  ===,!=== **Left**

  && **Left**

  || **Left**

  =,+=,-=,*=,/=,&&=,||=,%= **Right**

- Example of operator precedence and associativity

```
  var a=2, b=4,c,d;
C=3+a*b;
//* is first, so c is now 11
 d=b/a/2;
// associates left , so d is now 1
```

Paranthesis can be used to force any desired precedence.

```
(a+b)*c
```

Addition will be done before multiplication

5.2Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

5.3.Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5" x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

5.4 Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

6)Math Object

Math Object provides collection of properties of Number objects and methods that operate on Number objects. The math object has methods for the trignometric functions such as sin(for sine) and cos(for cosine) as well as commonly used math operations such as floor, round, max, min,

Floor-truncate a number , round- to round a number,max –returns the largest of two numbers.

- All of the math methods are referenced through the Math object.

e.g., Math.cos(x)

```
<script type="text/javascript">
     var a=5.6784;
alert("Floor of a="+Math.floor(a));
//o/p--5
alert("round of a="+Math.round(a));
//o/p--6
alert("max of 78,12,38="+Math.max(78,12,38));
//o/p--78
alert("min of 78,12,38="+Math.min(78,12,38));
//o/p--12
</script>
```

Number Object

Number Object includes a collection of useful properties that have a constant value.

- Some useful properties:
- MAX_VALUE- Largest representable number on the computer being used

MAX_VALUE, MIN_VALUE, NaN, POSITIVE_INFINITY, NEGATIVE_INFINITY, PI

These properties are referenced through Number

- e.g., Number.MAX_VALUE

Any arithmetic operation that results in an error (eg:division by zero) or produces a number that cannot be represented as a double precision floating point number such as a number that is too large creates overflow returns the value not a number which is displayed as NaN .

NaN is not == to any number, not even itself - Test for it with isNaN(x)

Number object has the method, toString()-converts number through which it is called to a string.

```
var price=427,str_price;
str_price=price.toString();
```

Eg:

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

var a='hello';

if(isNaN(a))

     {alert('ENTER Number value');}

     else

     {alert('Thank u for entering Number value');}

</script>
```

7) String Concatenation Operator

- *String catenation operator* +

  txt1 = "smitha";
  txt2 = "jacob";
  txt3 = txt1 + " " + txt2;

o/p→ smitha Jacob

txt1 = "What a very ";
txt1 += "nice day";

o/p→ What a very nice day

8) Implicit Type Conversion

Coercions-Catenation coerces numbers to strings.Numeric operators (other than +) coerce strings to numbers (if either operand of + is a string, it is assumed to be catenation).

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

Example

```
x = 5 + 5;
y = "5" + 5;
z = "Hello" + 5;
```

- The result of *x*, *y*, and *z* will be:

  10
  55
  Hello5

  "August" + 1977 results "August 1977"

If you add a number and a string, the result will be a string!

Eg: 7 * "3"

This operator is used only with numbers. Javascript attempt to convert it into number. If conversion succeeds the value of the expression will be 21.If the second operand were a string that could not be converted to a number such as "August" the conversion would produce NaN

9) Explicit Type Conversion

1. Use the **String** and **Number** constructors
2. Use **toString** method of numbers
3. Use **parseInt** and **parseFloat** on strings

var num=6;

var str_value=num.**toString()**; O/P ➔Result is "6"

Var v=1555;

Var s=**String**(v); O/P => "1555"

Number can also be converted to String by concatenating it with the empty string.

String can be explicitly converted to numbers using a Number constructor.

```
var number=Number(aString);
```

```
<script type="text/javascript">
var a='5555';
var num=6;
     var str_value=num.toString();
     alert("   String result"+6+str_value);
//o/p–"66"
     var number=Number(a);
     alert(10+number);
//o/p--5565
</script>
```

JavaScript Strings

A string is a series of characters like "John Doe". Strings are written with quotes.

Example

```
var carName = "Maruthi";   // Using double quotes
var carName = 'Toyota';   // Using single quotes

var answer = "It's alright";//Single quote inside double quote

var answer = "He is called 'Johnny'";// Single quotes inside double
quotes
var answer = 'He is called "Johnny" ';    // Double quotes inside
single quotes
```

10) String properties & methods:

String methods can always be used through string primitive values as if values were objects.

The string object includes one property and large collection of methods.

```
var str="George";

 var len = str.length;   //len→6

 charAt(position) e.g., str.charAt(3)//r

indexOf(string) e.g., str.indexOf('o')//2

substring(from, to)e.g., str.substring(2,4)//org

toLowerCase()  e.g., str.toLowerCase() //george
```

11) ***The* typeof *operator***

JavaScript **typeof** operator is used to find the type of a JavaScript variable. The **typeof** operator returns the type of a variable or an expression. Returns "number", "string", or "boolean" for Number, String, or Boolean, "undefined" for Undefined,"function" for functions, and "object" for objects and NULL . Objects do not have types. If the operand is a variable that has not been assigned a value 'typeof' produces 'undefined' .

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

typeof 0                   // Returns "number"
typeof 314                 // Returns "number"
typeof 3.14                // Returns "number"
typeof (3)                 // Returns "number"
typeof (3 + 4)             // Returns "number
```

Undefined and null are equal in value but different in type:

typeof undefined // undefined

typeof null // object

12) **The Date Object**

The Date object is a datatype built into the JavaScript language. Create one with the new operator & Date constructor (no params).

var today = new Date();

Local time methods of Date object:

toLocaleString – returns a string of the date

getDate – returns the day of the month

getMonth – returns the month of the year (0 – 11)

getDay – returns the day of the week (0 – 6)

getFullYear – returns the year

getTime – returns the number of milliseconds since January 1, 1970

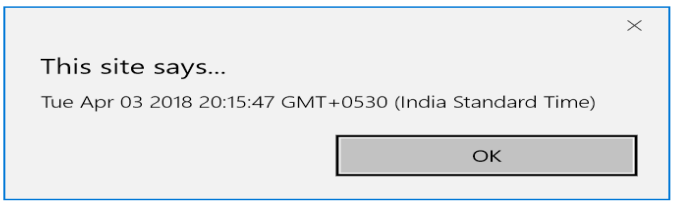
getHours – returns the hour (0 – 23)

getMinutes – returns the minutes (0 – 59)

getMilliseconds – returns the milliseconds (0 – 999)

//Date example

```
<!DOCTYPE html>
<html lang="EN">
<head><title>My Javascript page</title>
<script type="text/javascript">
<!--
var today=new Date();
 alert(today);
// -->
</script>
</head>
<body>
<h1>ICET</h1></body>
</html>
```




```
Example 2:

<!Doctype html>

<html>

<head>

<script>

var d= new Date();

var result =d.toLocaleString();

document.write("date and time as a string = " + result);

</script>

</head></body></html>
```

OUTPUT

```
time as a string = 10/4/2019, 10:28:17 AM
```

4.5 Screen Output and Keyboard Input

Every web page resides inside a browser window which can be considered as an object.An object of window is created automatically by the browser. The javascript model for the browser display window is the **Window** object. A *document* object represents HTML document. The model for the browser display window is the **Window** object. It represents the browser window.

The Window object has two properties, document and window, which refer to the **Document** and **Window** objects , respectively .

The **Document** object has several methods and properties. One method is **write**, which is used to create output.

The parameter is a string, often catenated from parts, some of which are variables
e.g., document.write("Answer: " + result + "
");

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML.

- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using **console.log()**.

Using innerHTML

- To access an HTML element use **document.getElementById(id)**
- The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

```
<!DOCTYPE html>
<html><body>
<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo">
   <script>
        document.getElementById("demo").innerHTML = 5 + 6;
   </script>

</p>
</body>
</html>
```

Using document.write()

document.write() for writing

Example

```
<!DOCTYPE html>
<html><body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script type = "text/javascript" >
     document.write(5 + 6);
</script>
</body>
</html>
```

The **Window** object has three methods for creating dialog boxes for user interactions , **alert, confirm , and prompt**

The **alert** method opens a dialog window and displays its parameter in that window. It also displays an **OK** button

```
alert("The sum is :"  +sum + "\n");
```

O/P

The **confirm** method opens a dialog window in which the method displays its string parameter along with two buttons **OK** and **Cancel** to offer the user the choice of continuing some process.

```
Var question = confirm("Do you want to continue this download");
```

O/P

The **prompt** method used to create dialog window that contains a textbox used to collect string of input from from the user .The window also includes 2 buttons: **OK** and **Cancel**.The prompt takes 2 parameters: ***the string*** that prompts user for input and a ***default string*** in case user does not type a string before pressing one of the 2 buttons.

```
name=prompt("What is your name?" ,"");
```

O/p

1.alert

Parameter in alert is plain text, not HTML.Opens a dialog box which displays the parameter string and an OK button.

```
<!DOCTYPE html>

<html lang ="en">
<head><title>hi.html</title>

<meta charset = "utf-8" />
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
   window.alert(5 + 6);
</script>
</body>
</html>
```

OUTPUT

My First Web Page

My first paragraph.

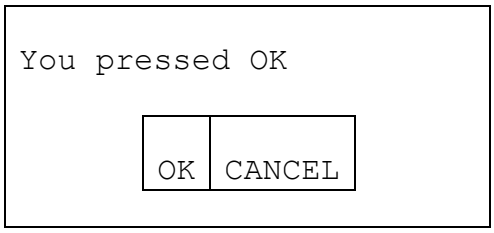
2.confirm

Opens a dialog box and displays the parameter and two buttons, OK and Cancel

```
if (confirm("Press a button!"))

 {
    txt = "You pressed OK!";
} else {
    txt = "You pressed Cancel!";

} window.alert(txt);
```



3.Prompt

Opens a dialog box and displays its string parameter, along with a text box and two buttons, OK and Cancel.

The second parameter is for a default response if the user presses OK without typing a response in the text box (waits for OK) .

```
var person = prompt("Please enter your name", "");
if (person == null || person == "") {
    txt = "User cancelled the prompt.";
} else {
    txt = "Hello " + person + "! How are you?";
}

window.alert(txt);
```

Sum of two numbers

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

var no1 = prompt("Please enter No1", "");

var no2 = prompt("Please enter No2", "");

if (no1 != "" || no2 != "")

 {

    var res = Number(no1)+Number(no2);

} else {

    res= "Failed ";
```

```
}
window.alert(res);
</script>
```

Example 2

```
<!DOCTYPE html>
<html lang = "en"><body><h1>Sum of Two numbers</h1>
<p>My first paragraph.</p>
<SCRIPT TYPE="text/javascript">
   var firstNumber,secondNumber,number1,number2,sum;
   firstNumber = window.prompt("Enter first integer", "" );
   secondNumber = window.prompt( "Enter second integer", "" );
   // convert numbers from strings to integers
   number1 = parseInt(firstNumber);
   number2 = parseInt( secondNumber );
   sum = number1 + number2;
   document.writeln( "<H1>The sum is " + sum + "</H1>" );
</SCRIPT></body></html>
```

Example 3:

Compute the real roots of a given quadratic equation

//roots.htm A document for roots.js

```
<!DOCTYPE html>
<html lang = "en">
  <head>
    <title> roots.html </title>
    <meta charset = "utf-8" />
  </head>
  <body>
    <script type = "text/javascript"  src = "roots.js" >
```

```
    </script>
  </body>
</html>
```

roots.js

//If the roots are imaginary, this script displays NaN, because that is what results from taking the square root of a negative number

```
var a = prompt("Find the solution for ax^2 + bx +c \n What is the value of 'a'? \n", "");

var b = prompt("What is the value of 'b'? \n", "");

var c = prompt("What is the value of 'c'? \n", "");

// Compute the square root and denominator of the result

var root_part = Math.sqrt(b * b - 4.0 * a * c);

var denom = 2.0 * a;

var root1 = (-b + root_part) / denom;

var root2 = (-b - root_part) / denom;

document.write("The first root is: ", root1, "<br />");

document.write("The second root is: ", root2, "<br />");
```

4.6 CONTROL STATEMENTS

Conditional Statements

- **if statement** - use this statement if you want to execute some code only if a specified condition is true
- **if...else statement** - use this statement if you want to execute some code if the condition is true and another code if the condition is false.
- **if...else if....else statement** - use this statement if you want to select one of many blocks of code to be executed .
- **switch statement** - use this statement if you want to select one of many blocks of code to be executed .

Control Statements-1

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

```
}
<script>
var x = prompt("Enter the Name \n", "");
if(x=="admin")
{
alert ("Welcome Administrator");
}
</script>
```

if-else

```
if (condition)
{
code to be executed if condition is true
}
else
{
code to be executed if condition is not true
}
```

Example

```
<script type = “text/javascript”>
var x = prompt("Enter the Number \n", "");
if(x<0)
{
alert ("negative");
}
else
{
alert ("positive");
}
```

</script>

else if Statement

Use the else if statement to specify a new condition if the first condition is false.

Syntax

if (*condition1*)

{
block of code to be executed if condition1 is true
} else if (*condition2*)

{
block of code to be executed if the condition1 is false and condition2 is true
} else {
block of code to be executed if the condition1 is false and condition2 is false
}

Example

```
<script>
var time = prompt("Enter the time \n", "");
if (time < 10)
 {
    greeting = "Good morning";
} else if (time < 20)
 {
    greeting = "Good day";
} else
 {
    greeting = "Good evening";
}
alert(greeting);
</script>
```

JavaScript Switch Statement

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

Syntax

switch(*expression*)

```
{
  case n:
    code block
    break;
  case n:
    code block
    break;
  default:
    code block
}
```

- 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 e

```
<script type="text/javascript">
var d= new Date();
Var day = d.getDay();
switch (day) {
    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";
    }
    alert(day);
    </script>
```

Loop Statements

Different Kinds of Loops

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

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

for loop

syntax:

```
for (statement 1; statement 2; statement 3)
{
    code block to be executed
}
```

- Statement 1 is executed before the loop (the code block) starts.
- Statement 2 defines the condition for running the loop (the code block).
- Statement 3 is executed each time after the loop (the code block) has been executed.

Example

```
for (i = 0; i < 5; i++)

 {
    text += "The number is " + i + "<br>";

alert(text);
}
```

for/in Loop

- The JavaScript for/in statement loops through the properties of an object:
- Example

```
var person = {fname:"John", lname:"Doe", age:25};
var text = "";
var x;
for (x in person)

 {
    text += person[x];
alert(text);}
```

While Loop

- The while loop loops through a block of code as long as a specified condition is true.

```
while (condition)

{
    code block to be executed
}
```

Example

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

var i=0;

while (i < 10)

 {

   alert("i="+i);

i++;

}

</script>
```

Do/While Loop

- The do/while loop is a variant of the while loop.
- This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {
    code block to be executed
}
while (condition);
```

Example

```
<script type="text/javascript">
var i=0;
do
 {
     alert("i="+i);
i++;
}while (i <= 3);
</script>
```

Break

The break statement "jumps out" of a loop. It was used to "jump out" of a switch() statement. break statement also be used to jump out of a loop. The break statement breaks the loop and continues executing the code after the loop (if any):

Example

```
<script type="text/javascript">
var i,text="";
for (i = 0; i < 10; i++)
{
    if (i === 3) { break; }
    text += "The number is " + i;
alert(text);
}
```

```
</script>
```

Continue Statement

- The continue statement "jumps over" one iteration in the loop.
- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- This example skips the value of 3:

```
<script type="text/javascript">
var i, text="";
for (i = 0; i < 10; i++) {
    if (i === 3) { continue; }
    text += "The number is " + i;
}alert(text);
</script>
```

4.7 OBJECTS CREATION

JavaScript Objects

- In JavaScript, object is the king. Almost "everything" is an object.
- Booleans can be objects (if defined with the new keyword)
- Numbers can be objects (if defined with the new keyword)
- Strings can be objects (if defined with the new keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects
- All JavaScript values, except primitives, are objects.
- JavaScript objects are written with curly braces.
- Object properties are written as name:value pairs, separated by commas.

Example

var person = {firstName:“xyz”, lastName:“pqr", age:50, eyeColor:"blue"};

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

A JavaScript object is a collection of named values . The named values, in JavaScript objects, are called properties.

Creating a JavaScript Object

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

1)Define and create a single object, using an object literal.

2)Define and create a single object, with the keyword new.

3)Define an object constructor, and then create objects of the constructed type.

Using an Object Literal

Easiest way to create a JavaScript Object. Using an object literal, define and create an object in one statement.

An object literal is a list of name:value pairs inside curly braces {}.

The following example creates a new JavaScript object with four properties:

```
var person = {
    firstName:“xyz",
    lastName:“pqr",
    age:50,
    eyeColor:"blue"
};
```

Using the JavaScript Keyword new

Most common way of creating javascript objects.

The following example also creates a new JavaScript object with four properties:

Example

```
var person = new Object();
person.firstName = “xyz";
person.lastName = “pqr";
person.age = 50;
person.eyeColor = "blue";
```

3)Define an object constructor, and then create objects of the constructed type

Because the objects can be nested , we can create new objects with the properties of person with the properties of its own.

person.address =new Object();

person.address.district="Ernakulam";

person.address.state="Kerala";

JavaScript Object Properties

Properties are the values associated with a JavaScript object.A JavaScript object is a collection of unordered properties.Properties can usually be changed, added, and deleted, but some are read only.

syntax for accessing the property of an object is:

objectName.property // person.age

or

objectName["property"] // person["age"]

or

objectName[expression] // x = "age"; person[x]

person.firstname + " is " + person.age + " years old.";

or

person["firstname"] + " is " + person["age"] + " years old.";

JavaScript for...in Loop

The JavaScript for...in statement loops through the properties of an object. The block of code inside of the for...in loop will be executed once for each property.

Syntax

for (variable in object)

{
code to be executed
}

Example

```
var person = {fname:"xyz", lname:"pqr", age:25};
for (x in person)

 {
   txt += person[x];
}
```

Adding New object Properties

You can add new properties to an existing object by simply giving it a value. Assume that the person object already exists - you can then give it new properties:

Example

```
person.city = "kochi";
```

Deleting Object Properties

- The delete keyword deletes a property from an object:
- **Example**

```
var person = {fname:"xyz", lname:"pqr", age:50, city:"kochi"};
delete person.age;   // or delete person["age"];
```

The delete keyword deletes both the value of the property and the property itself. After deletion, the property cannot be used before it is added back again. The delete operator is designed to be used on object properties. It has no effect on variables or functions.

Adding a Method to an Object

Adding a new method to an object is easy:

```
person.name = function ()

{
   return this.firstName + " " + this.lastName;
};
```

Strings, Numbers, and Booleans as Objects

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

```
var x = new String();

          // Declares x as a String object
var y = new Number();

      // Declares y as a Number object
var z = new Boolean();

    // Declares z as a Boolean object
```

Avoid String, Number, and Boolean objects.

4.8 JavaScript Arrays

- JavaScript arrays are used to store multiple values in a single variable.

- Example

var cars = [“Maruthi", “Toyoto", "BMW"];

- An array can hold many values under a single name,
- Access the values of array by referring to an index number.

Creating an Array

Using an array literal we can create a JavaScript Array.

Syntax:

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

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example

```
var s = new Array(“AA", ”BB", “CC”);
var list = new Array(2, 4, 6, 8);
```

Access the Elements of an Array

An array element can be accessed by referring to the index number.

To access the value of the first element in s:

var name = s[0];

EXAMPLE

```
<!DOCTYPE html>
<html>
<body>
<h1 id="demo"></h1>
<script>
var s = ["AA", "BB", "CC"];
document.getElementById("demo").innerHTML = s[0]+
" "+s[1]+" "+s[2];
</script>
</body>
```

```
</html>
```

Array Properties and Methods

1. The **length** property of an array returns the length of an array (the number of array elements). Ex)

var s = new Array("AA", "BB", "CC");

alert(s.length); //return 3

2. Adding Array Elements

- To add a new element to an array is using the push method:

Example

s.push("XX");

Example

```
<!DOCTYPE html>
<html><body><h1 id="demo"></h1>
<script>
var s = ["AA", "BB", "CC"];
s.push("XX");
for (i = 0; i < s.length; i++) {
document.getElementById("demo").innerHTML  +=
 "<ul><li>" + s[i] + "</li>";
}
</script>
</body></html>
```

- AA
- BB
- CC
- XX

Add new element to array

- New element can also be added to an array using the length property:

```
var s = ["BB", "OO", "AA", "MM"];

s[s.length] = "LL";

//add as last element
```

- Adding elements with high indexes can create undefined "holes" in an array:

```
var s = ["BB", "OO", "AA", "MM"];

s[6] = "PP";
```

Popping

- The pop() method removes the last element from an array:
- Example

```
var s = ["AA", "BB", "CC"];

s.pop();   // Removes the last element ("CC") from array s
```

- The pop() method returns the value that was "popped out":
- Example

```
var s = ["BB", "OO", "AA", "MM"];

var x = s.pop();      // the value of x is "MM"
```

Shifting Elements

- Shifting is equivalent to popping, working on the first element instead of the last.
- The shift() method removes the first array element and "shifts" all other elements to a lower index.
- Example

```
var s = ["BB", "OO", "AA", "MM"];
s.shift();          // Removes the first element "BB" from ' s'
```

unshift() method

- The **unshift()** method adds a new element to an array (at the beginning), and "shifts" older elements:

Example

```
var s = ["BB", "OO", "AA", "MM"];

s.unshift("AA");     // Adds a new element "AA" to s
```

- The unshift() method returns the new array length.

Example

```
var s = ["BB", "OO", "AA", "MM"]; alert(s.unshift("AA");    //
Returns 5
```

Deleting Elements

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete:

Example

var s = ["BB", "OO", "AA", "MM"]; delete s[0]; // Changes the first element in s to undefined

Using delete may leave undefined holes in the array. Use pop() or shift() instead.

Merging (Concatenating) Arrays

- **concat()** method creates a new array by merging (concatenating) existing arrays:
- Example (Merging Two Arrays)

var girls = ["ammu", "annu"];
var boys = ["Emil", "richu", "Linu"];
var mystds = girls.concat(boys); //Concatenates (joins) girls and boys

The concat() method can also take values as arguments:

Example (Merging an Array with Values)

var arr1 = ["Cecilie", "Lone"];
var myChildren = arr1.concat(["Emil", "asi", "Linu"]);

slice() method

slice() method slices out a piece of an array into a new array.

Eg: slices out a part of an array starting from array element 1 ("Orange"):

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
```

- slice() method creates a new array. It does not remove any elements from the source array.It returns the part of the array object specified by its parameters.
- This example slices out a part of an array starting from array element 3 ("Apple"):

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(3);
```

returns "Apple" ,"Mango"

- The slice() method can take two arguments like slice(1, 3).
- The method then selects elements from the start argument, and up to (but not including) the end argument.

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1, 3);
```

returns “Orange” ,”Lemon”

Sorting an Array

- The sort() method sorts an array alphabetically:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();          // Sorts the elements of fruits
```

By default, the sort() function sorts values as strings. This works well for strings . However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1". Because of this, the sort() method will produce incorrect result when sorting numbers. Use **compare function** for numeric sorting .

Reversing an Array

The reverse() method reverses the elements in an array. use it to sort an array in descending order:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();              // Sorts the elements of fruits
fruits.reverse();           // Reverses the order of the elements
```

Associative Arrays

Arrays with named indexes are called associative arrays (or hashes). JavaScript does not support arrays with named indexes. In JavaScript, arrays always use numbered indexes.

Difference Between Arrays and Objects

- Arrays use numbered indexes
- use arrays when you want the element names to be numbers.
- Javascript does not support associative arrays

- Arrays use numbered indexes
- use arrays when you want the element names to be numbers.
- Javascript does not support associative arrays

4.9 FUNCTIONS

```
function function_name([formal_parameters])

{

 -- body -

}
```

Return value is the parameter of return

- If there is no return, or if the end of the function is reached, undefined is returned

- If return has no parameter, undefined is returned

- Functions are objects, so variables that reference them can be treated as other object references

  - If fun is the name of a function,

```
function fun()
{document.write("This is good!<br />";}
  ref_fun = fun;
  fun();
 ref_fun();  /* A call to fun */
```

```
// function returns a value
<script>
function myFunction(a, b)
{
   return a * b;
}
var x = myFunction(4, 3);
alert(x);
</script>
```

```
// function returns a value
<script>
function myFunction(a, b)
{
   return a * b;
}
var x = myFunction(4, 3);
alert(x);
</script>
```

Parameters

The parameter values that appear in a call to a function are called actual parameters. The parameter names that appear in the header of a function definition, which correspond to the actual parameters in calls to the function, are called formal parameters. JavaScript uses the pass-by-value parameter-passing method.When a function is called, the values of the actual parameters specified in the call are copied into their corresponding formal parameters, which behave exactly like local variables.

JavaScript in <head> Example

```
<!DOCTYPE html>
<html><head>
<script>
```

```
function test() {

    document.getElementById("hello").innerHTML = "Welcome to Javascript
function.";

}

</script>

</head>

<body><p id="hello">Hello World</p>

<input type="button" onclick="test()" value="Try to change"/>

</body></html>
```

JavaScript in <body> Example

```
<!DOCTYPE html>
<html><body>
<h1>A Web Page</h1>
<p id="demo">Hello</p>
<input type="button" onclick="test()" value="change"/>

<script>
function test () {
   document.getElementById("demo").innerHTML = "Welcome to HTML function";
}
</script>
</body></html>
```

Array example using function

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

function fun1(my_list)

{

     alert(my_list);    //2,4,6,8

     var list2 = new Array(1, 3, 5);

     my_list[3] = 14;

     alert(my_list);  //2,4,6,14

     my_list = list2;

     alert(my_list);  //1,3,5

}
```

```
var list = new Array(2, 4, 6, 8);

fun1(list);

    </script>
```