Unit 2 – The JavaScript 10 hrs

Overview of JavaScript, Execution Environment, Object orientation and JavaScript, Syntactic characteristics, Primitives, operations, and expressions, Arrays, Functions, Pattern matching using regular expressions, Examples. Events and Event Handling.

Overview of the javascript :

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML
- Open and cross-platform.

Advantages of JavaScript or The merits of using JavaScript are:

JavaScript is dynamically typed if the type of a variable is interpreted at runtime. This means that you as a programmer can write a little quicker because you do not have to specify type every time.

Example: Perl

1. An interpreted language: javascript is an interpreted language, it doesn't require any compilation steps. This provides easy compilation steps.
2. Embedded within html :javascript doesn't require any separate or special editor for programmes.It can be simply typed in notepad and saved with .htm or .html extension.
3. Minimal syntax and easy to learn: by learning few commands and simple rules of syntax, whole application can be built.
4. Quick development: as there is no compilation time ,scripts can be developed in short time quickly. This includes GUI enhancement features.
5. Designed for simple and small programs: It is well suited for simple and small programs. i.e they can be easily written and executed as well as integrated into the web page.
6. Performance: javascript can be written such that they are quite small and compact. This minimizes storage requirement on the server.
7. Procedural capabilities: As in any other programming the looping structures are supported. So does the javascript provide syntax and procedural capabilities.
8. Designed for programming user events: javascript supports object/event based programming. It recognizes when the button is pressed and the event that has to occur. Mouse events in HTML can also be implemented in JavaScript ,like Mouseover, Onclick etc.
9. Easy debugging and testing: In javascript errors can be located line by line, and they are also listed as they encountered. Hence it is easy to debug errors.
10. Platform independence/architecture neutral : javascript is completely independent of the platform that it works. It is understood by any javascript enabled browser. Since the browser is with respect to specific platform, the javascript interpretation depends on the browser.

Limitations of JavaScript :

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features:

- 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 multithreading or multiprocessor capabilities. Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

Execution environment in javascript :

It is defined as the execution environment in which the javascript code is executed.

- All JavaScript variables are properties of some object. The properties of the Window object are visible to all JavaScript scripts that appear either implicitly or explicitly in the window's XHTML document, so they include all of the global variables.
- When a global variable is created in a client-side script, it is created as a new property of the Window object. There can more than one window object.
- Every Window object has a property named document, which is a reference to the Document object that the window displays.
- Every Document object has a forms array, each element of which represents a form in the document. Each forms array element has an elements array as a property, which contains the objects that represent the XHTML form elements, such as buttons and menus.
- Document objects also have property arrays for anchors, links, images, and applets. There are many other objects in the object hierarchy below a Window Object.

Object orientation and javascript :

- JavaScript is NOT an object-oriented programming language.
- Does not support class-based inheritance it has **prototype-based** inheritance, which is much different.
- Cannot support polymorphism. As the classes are not defined here.
- JavaScript objects are collections of *properties*, which are like the members of classes in Java and C++.
- JavaScript has primitives for simple types. Which includes integers and float values.
- The root object in JavaScript is **Object** – all objects are derived from **Object.**
- All JavaScript objects are accessed through references .

Javascript object:

- In javascript ,objects are collection of properties. Each property is either a data property or a function Or method property.
- Data properties appear in two categories.

a)*Primitive values* : JavaScript use non-object types for simplest data types called **primitives**. This helps faster operation of values.

b)*References to other objects* : In JavaScript variables that refer to an object are often called ***objects*** rather than ***references.***

- All objects in a JavaScript program are indirectly accessed through variables. Such a variable is like a reference in Java. All primitive values in JavaScript are accessed directly. These are often called value types.
- 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 the most generic of all objects, having some methods but no data properties

General Syntactic characteristics Javascript :

- The JavaScript can be Scripts can appear directly as the content of a <script> tag. The type attribute of <script> must be set to "text/javascript". The JavaScript script can be indirectly embedded in an XHTML document with the src attribute of a <script> tag, whose value is the name of a file that contains the script, for example

```
//external javascript
<script type = "text/javascript" src= "xxx.js">          </Script>
Or
//internal jaavscript
<html>
<body>
<script type = "text/javascript">
document.write("Hello World!");
</script>
</body>
</html>
```

Indirect method of embedding JavaScript in XHTML document has the advantage of hiding the script from the browser user.

- **Identifiers:** they are nothing but names given to objects or variables.They must begin with a letter, an underscore (_), or a dollar sign ($). Subsequent characters may be letters, underscores, dollar signs, or digits. There is no length limitation for identifiers.
- **Reserved words**:they are nothing but the keywords which can't be used for other purposes ,as variable names or identifiers. There exists 25 main reserved words in javascript. Which are depicted in the following table?

break	delete	function	return	typeof
case	do	if	switch	var
catch	else	in	this	void
continue	finally	instanceof	throw	while
default	for	new	try	with

Table 2.1 :javascript reserved words.

- JavaScript has a large collection of predefined words, including alert, open, java, and self.
- Sometimes Semicolon (;) is Optional. Normally placed at the end of the js statement. Comments can be as follows.

  //Single Line Comment

/* Multiple line Comment */

// dynamically changing properties

JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment. The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

- There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows:
- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

Primitives:

- JavaScript uses non-object types for some of its simplest types; these non-object types are called primitives. Which can be directly interpreted unlike objects. JavaScript has 5 primitive types:
  - Number
  - String
  - Boolean
  - Undefined
  - Null
- **Number:** These are nothing but numerical values basically includes integer and float types. Floating point literals can have decimal or exponent or both Exponent are specified in uppercase or lowercase **e** and a possibly integer literal .

  Example: **72 7.2 .72 72. 7e2 .7e2 7.e2 7.2E-2**
- **String:** A string literal Delimited by single quote(‘) or double quote(“).All string literals are primitive values. They are not an array of characters. There exists escape characters require backslash(\) e.g. \t
- **Boolean:** Boolean by has two values true\false. Automatically these are converted to 0 and 1.
- Number String, Boolean(True/False) objects are called **wrapping object** because they provide properties and methods that are for values of primitive types
- **Undefined:** If any variable is explicitly defined but not assigned a value, Undefined will be displayed.
- **Null:** The only value of type Null is the reserved word null, which indicates no value. A variable is null if it has not been explicitly declared or assigned a value.
- **Literal values:**they are nothing but the values given to the variables.

Operators and expressions:

The variables can be declared in different forms. The keyword var is used for declaring all kinds of variables. **Example:** var index,

Pi=3.14,

Car=”autogear”,

Flag=true;

- The variables are dynamically typed.Any variable can be used as two things.Primitive value or object reference.
- The interpreter determines the type of a particular occurrence of a variable.Variables can be either implicitly or explicitly declared

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

- **Variable scoping**: Local variables are the ones ,which are declared within the function using var keyword. without var it becomes global variable

Operators:

An operator is used to transform one or more values into single resultant value.the values to which operands applied is referred to as operands.

The combination of the operator and operators is an expression.

There exists several operators in javacript, as in other languages like c++ ,java etc.they are as follows. Increment(**++),**decrement(**--),**unary plus(**+),**unary minus**(-),** multiplication(***),**division(/ **),**modulus(**%)**. All operations are in double precision (64 bits data consumption).

- It follows same precedence and associatively as Java.
- The associativity rules of a language specify which operator is evaluated first when two operators with the same precedence are adjacent in an expression. when multiple operators of same precedence in single statement occurs.
- Increment/decrement operator implies postfix or prefix methods as in other languages.

Operator	Associativity
++, --, unary -, unary +	Right (though it is irrelevant)
*, /, %	Left
Binary +, binary -	Left

The first operators listed have the highest precedence.

Example of operator precedence and associativity, consider the following code:

```
var a = 2,
    b = 4,
    c,
    d;
c = 3 + a * b;
// * is first, so c is now 11 (not 24)
d = b / a / 2;
// / associates left, so d is now 1 (not 4)
```

- Parentheses can be used to force any desired precedence. For example (a+b)*c.
- **String operator (+):**Strings in javascript are unit scalar values. String catenation is specified with the operator denoted by a plus sign (+). For example, if the value of first is “Freddie”, the value of the following expression is “Freddie Freeloader”:
Example: first + “ Freeloader”.

Arithmetic operators:

Sl no	Operator and description
1	+ (Addition) Adds two operands Ex: A + B will give 30
2	- (Subtraction) Subtracts the second operand from the first Ex: A - B will give -10
3	* (Multiplication) Multiply both operands Ex: A * B will give 200
4	/ (Division) Divide the numerator by the denominator Ex: B / A will give 2
5	% (Modulus) Outputs the remainder of an integer division Ex: B % A will give 0
6	++ (Increment) Increases an integer value by one Ex: A++ will give 11
7	-- (Decrement) Decreases an integer value by one Ex: A-- will give 9

Following example shows the use of the arithmetic operators:

```
<html>
<body>
<script type="text/javascript">
<!--
var a = 33;
var b = 10;
var c = "Test";
var linebreak = "<br />";
document.write("a + b = ");
result = a + b;
document.write(result);
document.write(linebreak);
document.write("a - b = ");
result = a - b;
document.write(result);
document.write(linebreak);
document.write("a / b = ");
result = a / b;
document.write(result);
document.write(linebreak);
document.write("a % b = ");
result = a % b;
document.write(result);
document.write(linebreak);
document.write("a + b + c = ");
result = a + b + c;
document.write(result);
document.write(linebreak);
a = a++;
document.write("a++ = ");
result = a++;
document.write(result);
```

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

Output:
a + b = 43 a - b = 23 a / b = 3.3 a % b = 3 a + b + c = 43Test a++ = 33 b-- = 10
Set the variables to different values
```

Logical operators:

These are used for performing the operations on the boolean values.

Sl no	Operator with description
1	&& (Logical AND) If both the operands are non-zero, then the condition becomes true. Ex: (A && B) is true.
2	|| (Logical OR) If any of the two operands are non-zero, then the condition becomes true. Ex: (A || B) is true.
3	! (Logical NOT) Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. Ex: ! (A && B) is false.

Consider the following example:

```
<html>
<body>
<script type="text/javascript">
<!--
var a = true;
var b = false;
var linebreak = "<br />";
document.write("(a && b) => ");
result = (a && b);
document.write(result);
document.write(linebreak);
document.write("(a || b) => ");
result = (a || b);
document.write(result);
document.write(linebreak);
document.write("!(a && b) => ");
result = (!(a && b));
document.write(result);
document.write(linebreak);
//-->
</script>
```

```
Output:
(a && b) => false (a || b) => true !(a && b) => true
```

Assignment operators:

Assignment operators are used for updating the values of the variables. following examples depict the operators.

Sl no	Operator and description
1	**= (Simple Assignment)** Assigns values from the right side operand to the left side operand . **Ex:** C = A + B will assign the value of A + B into C
2	**+= (Add and Assignment)**It adds the right operand to the left operand and assigns the result to the left operand. **Ex:** C += A is equivalent to C = C + A
3	**-= (Subtract and Assignment).**It subtracts the right operand from the left operand and assigns the result to the left operand. **Ex:** C -= A is equivalent to C = C - A
4	***= (Multiply and Assignment)** It multiplies the right operand with the left operand and assigns the result to the left operand. **Ex:** C *= A is equivalent to C = C * A
5	**/= (Divide and Assignment).**It divides the left operand with the right operand and assigns the result to the left operand. **Ex:** C /= A is equivalent to C = C / A
6	**%= (Modules and Assignment).**It takes modulus using two operands and assigns the result to the left operand. **Ex:** C %= A is equivalent to C = C % A

```
<html>
<body>
<script type="text/javascript">
<!--
var a = 33;
var b = 10;
var linebreak = "<br />";
document.write("Value of a => (a = b) => ");
result = (a = b);
document.write(result);
document.write(linebreak);
document.write("Value of a => (a += b) => ");
result = (a += b);
document.write(result);
document.write(linebreak);
document.write("Value of a => (a -= b) => ");
result = (a -= b);
document.write(result);
document.write(linebreak);
document.write("Value of a => (a *= b) => ");
result = (a *= b);
document.write(result);
document.write(linebreak);
```

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

Output:

Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0

Comparision operators:

Sl no	Operator and description
1	**== (equal)** .Checks if the value of two operands are equal or not, if yes, then the condition becomes true. **Ex:** (A == B) is not true.
2	**!= (Not Equal)**.Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. **Ex:** (A != B) is true.
3	**> (Greater than)**.Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A > B) is not true.
4	**< (Less than)** Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A < B) is true.
5	**>= (Greater than or Equal to)** Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A >= B) is not true.
6	**<= (Less than or Equal to)** .Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A <= B) is true.

Consider the following example:

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

```
var b = 20;
var linebreak = "<br />";
document.write("(a == b) => ");
result = (a == b);
document.write(result);
document.write(linebreak);
document.write("(a < b) => ");
result = (a < b);
document.write(result);
document.write(linebreak);
document.write("(a > b) => ");
result = (a > b);
document.write(result);
document.write(linebreak);
write("(a != b) => ");
result = (a != b);
document.write(result);
document.write(linebreak);
document.write("(a >= b) => ");
result = (a >= b);
document.write(result);
document.write(linebreak);
document.write("(a <= b) => ");
result = (a <= b);
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and different operators and then
try...</p>
</body>
</html>
 Output:
```

(a == b) => false (a < b) => true (a > b) => false (a != b) => true (a >= b) => false (a <= b) => true.

The conditional expression or ternary operator:

The conditional or ternary operator is used for if-else type condition checking. which has the syntax as shown below.

Condtion? Value1 :value2

The condition is checked first, if it's true the value1 is returned else the value2 is returned.

Special operators:

1. delete operators: The delete operator is used to delete the property of the object or an element of an array at the index. Ex: delete myArray[5].//deletes sixth element.

2. new operators: the new operator is used to create the instance of an object type. Ex: myArray=new Array().

3. void operator: the void operator does not return a value. It is typically used in javascript to return a URL with no values.

Type casting:

Implicit type conversions:

- When one primitive type is operated with the other primitive type ,javascript implicitly converts to the preferred primitive type, this is called implicit type conversion.
- The JavaScript interpreter performs several different implicit type conversions. Such conversions are called **coercions.**
- If either operand of a + operator is a string, the operator is interpreted as a string catenation operator. If the other operand is not a string, it is coerced to a string.

For example, consider the following expression:

1) August " + 1977 => "August 1997"
2) 7*"3" => 21 //second value 3 is string but coerced as numerical value.

Explicit Type Conversions:

There are several different ways to force type conversions, primarily between strings and numbers. Strings that contain numbers can be converted to numbers with the String constructor, as in the following code:

var str_value = String(value);

example:

a)var num = 6;
var str_value = num.toString(); //converts number to string
var str_value_binary = num.toString(2); // converts the binary value to string

In the first conversion, the result is "6"; in the second, it is "110".

b)parseInt("34") ---> 34
parseInt("34xyz") ---> 34
parseFloat("3.4xyz") ---> 3.4
parseInt("xyz34") ---> NaN .

Functions (built-in) used in javscript:

JavaScript uses certain built in functions such as ,pasreInt(),parseFloat(),eval(),string(),isNaN() etc

a)The parseInt function: It searches its string parameter for an integer literal. If one is found at the beginning of the string, it is converted to a number and returned. If the string does not begin with a valid integer literal, NaN is returned.

Example : The parseInt() function parses a string and returns an integer.

- <script type="text/javascript">
  document.write(parseInt("10")+"
");
  document.write(parseInt("10.33")+"
");
  document.write(parseInt("34 45 65)+"
");
  document.write(parseInt(" 60 ") + "
");
  document.write(parseInt("40 years") + "
");
  document.write(parseInt("He was 40") + "
");
  </script>
- O/p:10 10 34 60 40 NaN

b)The parseFloat function: It is similar to parseInt, but it searches for a floating-point literal, which could have a decimal point, an exponent, or both. In both parseInt and parseFloat, the numeric literal could be followed by any nondigit character without causing any problem.

Example:document.write(parseFloat("3.4xyz")+"
");---> 3.4

document.write (parseFloat("he was 3.4") +"
");-→NaN

document.write(parseFloat("3.4 years")+"
"); →3.4

c) isNaN : It stands for Not a Number. E.g. isNaN() (Devide by Zero error)

- An arithmetic operation that creates overflow returns NaN.
- **NaN** is not == to any number, not even itself.we can test using isNaN(x)
- **Number** object has the method, **toString** Conversions from strings to numbers, that do not work return **NaN** (Not a number).
- when number computation fails e.g. Overflow

d) eval() : evaluate function can be used to convert a string expression to numerical -value.

Example: the following results in the value 105 being assigned to the variable grand_total.

var grand_total=eval("10*10+5);

e) string(): converts an object's value to the string. it has the syntax: string (object).

Example: var x1=bollean(0);// o/p is false

var x2=new date();// o/p is sun jul 22 2018 11:42:19 GMT+0530(Indian standard time)

var x3= "1234"//o/p is 1234

The typeof Operator:

The **typeof** operator returns the type of its single operand.

- typeof produces **"number", "string",** or **"boolean".** If the operand is of primitive type Number, String, or Boolean, respectively.
- The operand for typeof can be placed in parentheses, making it appear to be a function. Therefore, typeof x and typeof(x) are equivalent.

---

optional----------

Selection statements:

The selection statements (if-then and if-then-else) of JavaScript are similar to those of the common programming languages. Either single statements or compound statements can be selected—for example,

```
if (a > b)
     document.write("a is greater than b <br />");
else {
     a = b;
     document.write("a was not greater than b <br />",
                    "Now they are equal <br />");
}
```

The switch statement:

JavaScript has a switch statement that is similar to that of Java. In any case segment, the statement(s) can be either a sequence of statements or a compound statement. The break statement transfers control out of the compound statement in which it appears.

The control expression of a switch statement could evaluate to a number, a string, or a Boolean value. Case labels also can be numbers, strings, or Booleans, and different case values can be of different types.

```
switch (expression) {
        case value_1:
                // statement(s)
        case value_2:
                // statement(s)
        ...
        [default:
                // statement(s)]
}
```

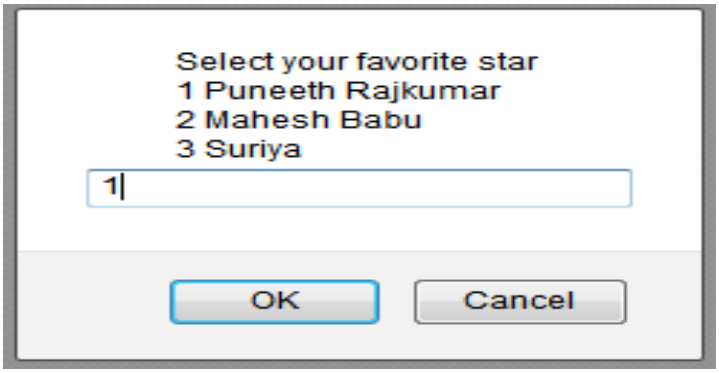
Consider the following example:

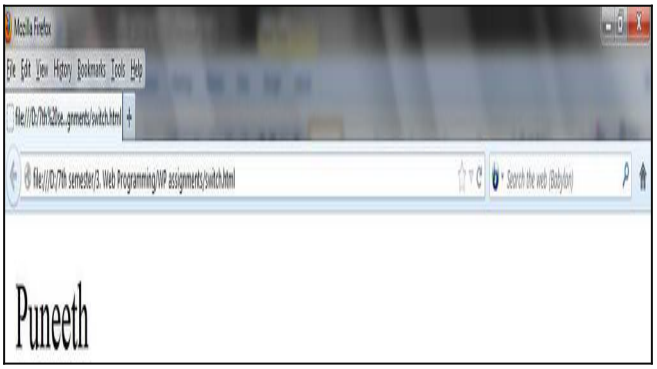
```
//switch.js
var choice = prompt("Select your favorite star \n" +
        "1 Puneeth Rajkumar \n" +
        "2 Mahesh Babu \n" +
        "3 Suriya \n");
switch(choice) {
case "1": document.write("Puneeth"); break;
case "2": document.write("Mahesh Babu"); break;
case "3": document.write("Suriya"); break;
default: document.write("invalid choice");
}
```

switch.html

```
<html>
<body>
<script type = "text/javascript" src = "switch.js">
</script>
</body>
</html>
```

Output:





Loop statements:

The javascript statements are similar to those that of c and java loop statements:

```
While(control expression)
{
single or compound statements;
```

}
For(initial expression: control expression: increment expression)
{
Statements;
}

----------optional topic
end----------

Screen input and output:

- The JavaScript model for the HTML document is the **Document object.**The model for the browser display window is the **Window object.**
- The Window object has two properties, **document** and **window**, which refer to the Document and Window objects, respectively.
- The Document object has a method, **write**, which dynamically creates content.The parameter is a string, often catenated from parts, some of which are variables.
  e.g., **document.write("Answer: " + result +"
");//Answer:42**
- The parameter is sent to the browser,which can be anything that can appear in an HTML document (value followed by
, but not \n).

Window includes three methods that create dialog boxes for three specific kinds of user interactions. these methods need not include an object reference. The three methods—**alert, confirm, and prompt.**

1.**Alert dailog box:**

Example:alert("Hey! \n");

1. The string parameter is not the XHTML code ,but it is a plain text.
2. It opens a dialog box which displays the parameter string and an OK button.
3. It waits for the user to press the OK button.
4. **alert** (“The sum is “ + sum + “\n”);

code :
```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called with the onload event");
}
</script>
</head>
<body onload="message()">
</body>
</html>
```

2. **confirm dailog box: confirm("Do you want to continue?");**
   1.opens a dialog box and displays the parameter and two buttons, OK and Cancel.
   2.It Returns a Boolean value, depending on which button was pressed (it waits for one/true).
   3.var question = **confirm** (“ Do you want to continue ?”);

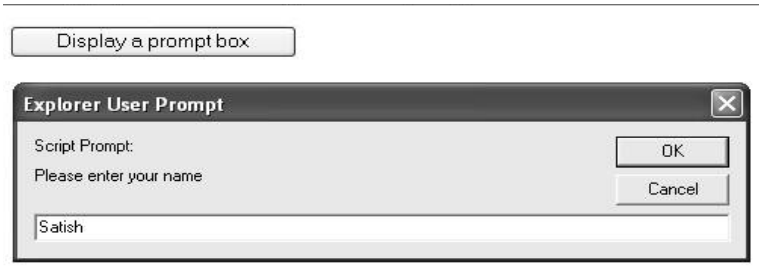
Code: <html> <head>

```
<script type="text/javascript">
var txt=“ "
function message() {
try { addalert("Welcome guest!“);   }
catch(err)
{
 txt="There was an error on this page.\n\n“;
 txt+="Click OK to continue viewing this page,\n“;
txt+="or Cancel to return to the home page.\n\n“;
if(!confirm(txt))
        {document.location.href=http://www.pes.edu/;}
            }
</script>  </head>  <body> </body> </html>
```

3.**Prompt window:**

It is similar to confirm dailog box.for example consider,prompt("What is your name?", "");

- 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**)
- name = **prompt** (“What is your name? “, “ “);
  - **roots.html root.js**



Object creation and modification in javascript:

Objects are often created with a **new** expression,which must include a call to a constructor method.The constructor that is called in the **new** expression creates the properties, that characterize the new object.

- In JavaScript however, the **new** operator creates a blank object - that is, object with no properties. The following statement creates an object that has no properties:

  var my_object = new Object();
- In this case, the constructor called is that of Object, which endows the new object with no properties, although it does have access to some inherited methods.
- The variable my_object references the new object.Calls to constructors must include parentheses, even if there are no parameters**.**
- The properties of an object can be accessed with dot notation, in which the first word is the object name and the second is the property name. Because properties are not variables, they are never declared. var my_object = new Object();
- At any time during interpretation, properties can be added to or deleted from an object. A property for an object is created by assigning a value to that property's name. Consider the following example:

  var my_car = {make: "Ford", model: "Contour SVT"};
- Properties can be accessed in two ways.

  var prop1 = my_car.make;

  var prop2 = my_car["make"];

  The variables prop1 and prop2 both have the value "Ford".

Arrays:

- In JavaScript,arrays are objects that have some special functionality.Array elements can be primitive values or references to other objects, including other arrays.JavaScript arrays have dynamic lengths.

  Example:
```
var my_list = new Array(1, 2, "three", "four");
var your_list = new Array(100);
```

In the first declaration, an Array object of length 4 is created and initialized. Notice that the elements of an array need not have the same type.

In the second declaration, a new Array object of length 100 is created, without actually creating any elements.

- The lowest index of every JavaScript array is zero. Access to the elements of an array is specified with numeric subscript expressions placed in brackets. The length of an array is the highest subscript to which a value has been assigned, plus 1.

  Example: **my_list[47] = 2222;**

For above example, if my_list is an array with four elements and the following statement is executed.the new length of my_list will be 48.

- The length of an array is both read and write accessible through the length property, which is created for every array object by the Array constructor. For example, **my_list.length = 1002;** An array is lengthened by setting its length property to a larger value, shortened by setting its length property to a smaller value.
- The below example, insert_names.js, illustrates JavaScript arrays. This script has an array of names, which are in alphabetical order. It uses prompt to get new names, one at a time, and inserts them into the existing array. Notice that each new name causes the array to grow by one element.

Dense arrays:

Dense array is an array, that has been created with each other of its elements being assigned as specific value. It is initializing array elements with the values.

Example: arrayName =new Array(value0,value1,…..value n)

A two-dimensional array: A two-dimensional array is implemented in JavaScript as an array of arrays. This can be done with the new operator or with nested array literals, as shown in the script nested_arrays.js:

```
// nested_arrays.js
//    An example illustrating an array of arrays

// Create an array object with three arrays as its elements
var nested_array = [[2, 4, 6], [1, 3, 5], [10, 20, 30]
                   ];

// Display the elements of nested_list
for (var row = 0; row <= 2; row++) {
  document.write("Row ", row, ":  ");

  for (var col = 0; col <=2; col++)
    document.write(nested_array[row][col], " ");

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

shows a browser display of nested_arrays.js.

Array Methods:

Array has several methods implemented along with it in javascript.some are described below.

1.The ***join*** method converts all of the elements of an array to strings and catenates them into a single string. If no parameter is provided to join, the values in the new string are separated by commas. If a string parameter is provided, it is used as the element separator.

Example: var names=new array [“mary”, “murray”,”murphy”, “max”]

var name_string= names**.**join(“:”);

//The value of name_string is now “Mary : Murray : Murphy : Max”.

2.The ***reverse*** method does what you would expect: It reverses the order of the elements of the Array object through which it is called.

3.The ***sort*** method coerces the elements of the array to become strings if they are not already strings and sorts them alphabetically.

Example: consider the following statement: names.sort(); //output [“Mary”, “Max”, “Murphy”, “Murray”].

4.The ***concat*** method catenates its actual parameters to the end of the Array object on which it is called. Thus, in the code can be given as,

var names=new array[“mary”,”murray”,”murphy”,”max”];

var new_names=names.concat(“moo”,”meo”);// o/p will be - mary,murray,murphy,max,moo,meo.

5.The ***slice*** method returning the part of the Array object specified by its parameters, which are used as subscripts. from the first parameter up to, but not including, the second parameter.

For example:1) var list=[2,4,6,8,10]

var list2=list.slice(1,3) //o/p [4,6] it will slice 1to 3 elements except for 1 and 3.

2)var list=[“jill”,”jack”,”jim”,”dill”];

var list2=list.slice(2)// o/p [“jim”,”dill”]

6. When the ***toString*** method is called through an Array object, each of the elements of the object is converted (if necessary) to a string. These strings are catenated, separated by commas.

7. The ***push, pop, unshift****, and* ***shift*** methods of Array allow the easy implementation of stacks and queues in arrays. The pop and push methods respectively remove and add an element to the high end of an array, as in the following code: var list= [“dasher”,”usher”,”dollar”,”blitzer”];

var der=list.pop;//it will remove blitzer.

der.push(“blitzer”); //it will add blitzer back to the list.

8. The ***shift*** and ***unshift*** methods respectively remove and add an element to the beginning of an array. For example, assume that list is created as before, and consider the following code:

var deer = list.shift(); // deer is now “Dasher”

list.unshift(“Dasher”); // This puts “Dasher” back on list

User defined functions:

1.Functions are declared and created using ***function*** keyword. It should have the following,

a) A name for the function.

b) A list of parameters(arguments) that will accept values passed to the functions when called.

c) A block of javascript code that defines what th functions does.

Syntax: function function_name(parameter1, parameter2,……parameter n)

{//block of code//

}

2.Function can be declared anywhere in the HTML page but preferably in <head></head> part. A **return** statement returns control from the function, in which it appears to the function’s caller.A function body may include one or more **return** statements. If there are no return statements in a function or if the specific return that is executed does not include an expression, the value returned is **undefined**.

fun1(); //returns an undefined value

result = fun2();

3. JavaScript functions are objects, so variables that reference them can be treated as are other object references.They can be passed as parameters, be assigned to other variables, and be the elements of an array. Because JavaScript functions are objects, their references can be properties in other objects, in which case they act as methods.

Example: function fun( )

{ document.write(“this is fun”); }

ref_fun()=fun(); //now ref_fun() refers to the fun() object

fun(); // a call to fun

ref_fun(); //also a call to a fun

Local variables:

- A variable declared (using **var**) within a JavaScript function becomes **LOCAL** and can only be accessed from within that function. (the variable has local scope).
- You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.
- Local variables are deleted as soon as the function is completed.
- The scope of a variable is the range of statements over which it is visible.

Global variables:

Variables declared outside a function, become **GLOBAL**, and all scripts and functions on the web page can access it.

- The Lifetime of JavaScript Variables **:**
- The lifetime JavaScript variables starts when they are declared.
- Global variables are deleted when you close the page.
- Assigning Values to Undeclared JavaScript Variables **:**
- If you assign a value to variable that has not yet been declared, the variable will automatically be declared as a GLOBAL variable.
- This statement: **carname="Volvo";**
  will declare the variable carname as a global variable , even if it is executed inside a function.

.

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, in effect, copied into their corresponding formal parameters, which behave exactly like local variables.
- Because of JavaScript's dynamic typing, there is no type checking of parameters. The called function itself can check the types of parameters with the ***typeof*** operator.
- The number of parameters in a function call is not checked against the number of formal parameters in the called function.
- In the function, excess actual parameters that are passed are ignored; excess formal parameters are set to ***undefined.***
- All parameters are communicated through a property array, arguments, that, like other array objects, has a property named ***length.*** By accessing arguments. length, a function can determine the number of actual parameters that were passed.

Recursive functions:

Recursive function refers to a situation where the functions calls themselves. Javascript also implements these types of functions.

Example: factorial function:

```
Function factorial(number){
If(number>1){
  return number* factorial(number-1);
}
 else{
return number;
}
}
```

Pattern matching using regular expressions:

1. A regular expression is an object that describes a pattern of characters.When we search in a text, we can use a pattern to describe ,what we are searching for.
2. A simple pattern can be one single character.A more complicated pattern can consist of more characters, and can be used for parsing, format checking, substitution and more.
3. Regular expressions are used to perform powerful pattern-matching and "search-and-replace" functions on text.
4. JavaScript has powerful pattern-matching capabilities based on regular expressions**.**
5. There are ***two approaches*** to pattern matching in JavaScript:

- one that is based on the ***RegExp object*** and
- one that is based on methods of the ***String object***.

6. The simplest pattern-matching method is **search**, which takes a pattern as a parameter.
7. The **search method** returns the **position in the String object** (through which it is called) at which the pattern matched. ***If there is no match, search returns –1.*** Most characters are normal, which means that, in a pattern,they match themselves.***The position of the first character in the string is 0.***

Consider the following example:

```
var str = "Rabbits are furry";
var position = str.search(/bits/);
if (position >= 0)
    document.write("'bits' appears in position", position,
                   "<br />");
else
    document.write("'bits' does not appear in str <br />");
```

produce the following output:

```
'bits' appears in position 3
```

Character and character class patterns:

1. Metacharacters are characters that have special meanings in some contexts in patterns.

· The following are the pattern metacharacters:

\ | ( ) [ ] { } ^ $ * + ? .

· Metacharacters can themselves be matched by being immediately preceded by a backslash.

· A period(.) matches any character except newline.
· Example: **/snow./ matches "snowy", "snowe", and "snowd"**
· Example: **/3\.4/ matches 3.4.** ***but /3.4/ would match 3.4 and 374, among others.***
· Example: **[abc] matches 'a' , 'b' & 'c'**
· Example: **[a-h] matches any lowercase letter from 'a' to 'h'**
· Example: **[^aeiou] matches any lowercase letter except 'a', 'e', 'i', 'o' & 'u'**

Name	Equivalent Pattern	Matches
\d	[0-9]	A digit
\D	[^0-9]	Not a digit
\w	[A-Za-z_0-9]	A word character (alphanumeric)
\W	[^A-Za-z_0-9]	Not a word character
\s	[\r\t\n\f]	A white-space character
\S	[^ \r\t\n\f]	Not a white-space character

.The following examples show patterns that use predefined character classes:

```
/\d\.\d\d/      // Matches a digit, followed by a period,
                // followed by two digits
/\D\d\D/        // Matches a single digit
/\w\w\w/        // Matches three adjacent word characters
```

.As another example, the pattern /[A-Za-z]\w*/ matches the identifiers (a letter, followed by zero or more letters, digits, or underscores) in some programming languages.

Anchors:

1.A pattern is tied to a string position with an anchor. A pattern can be specified to match only at the beginning of the string by preceding it with a circumflex (^) anchor.

For example, the following pattern matches "pearls are pretty" but does not match "My pearls are pretty": /^pearl/

2. A pattern can be specified to match at the end of a string only by following the pattern with a dollar sign anchor.

For example, the following pattern matches "I like gold" but does not match "golden": /gold$/

· Anchor characters are like boundary-named patterns:

3.They do not match specific characters in the string; rather, they match positions before, between, or after characters.

Pattern modifiers:

1.The modifiers are specified as letters just after the right delimiter of the pattern.
· The **i modifier** makes the letters in the pattern match either uppercase or lowercase letters in the string.
· For example, the pattern /Apple/i matches 'APPLE', 'apple', 'APPle', and any other combination of uppercase and lowercase spellings of the word "apple."
· The **x modifier** allows white space to appear in the pattern.

```
/\d+              # The street number
\s                # The space before the street name
[A-Z][a-z]+       # The street name
/x
```

is equivalent to

```
/\d+\s[A-Z][a-z]+/
```

Other pattern matching methods for strings:

- The replace method is used to **replace substrings of the String object** that match the given pattern.The **replace method** takes two parameters:
  the pattern and the replacement string**.**
- The **g modifier** can be attached to the pattern if the replacement is to be global in the string, in which case the replacement is done for every match in the string.
- The matched substrings of the string are made available through the predefined variables $1, $2, and so on. For example, consider the following statements:
  var str = “Fred, Freddie, and Frederica were siblings”;
  str.replace(/Fre/g, “Boy”);
- In this example, str is set to “Boyd, Boyddie, and Boyderica were siblings”, and $1, $2, and $3 are all set to “Fre”.

The **match method** is the most general of the String pattern-matching methods.

· The **match method** takes a **single parameter**: a pattern. It returns an array of the results of the pattern-matching operation.

· If the pattern has the **g modifier**, the returned array has all of the substrings of the string that matched.

· If the pattern does not include the **g modifier**, the returned array has the match as its first element, and the remainder of the array has the matches of parenthesized parts of the pattern if there are any:

```
var str =
  "Having 4 apples is better than having 3 oranges";
var matches = str.match(/\d/g);
```

In this example,matches is set to [4, 3**].**

The **split method** of String splits its object string into substrings on the basis of a given string or pattern. The substrings are returned in an array.

For example, consider the following code:

var str = “grapes:apples:oranges”;

var fruit = str.split(“:”);

In this example, fruit is set to [grapes, apples, oranges**].**

Errors in scripts:

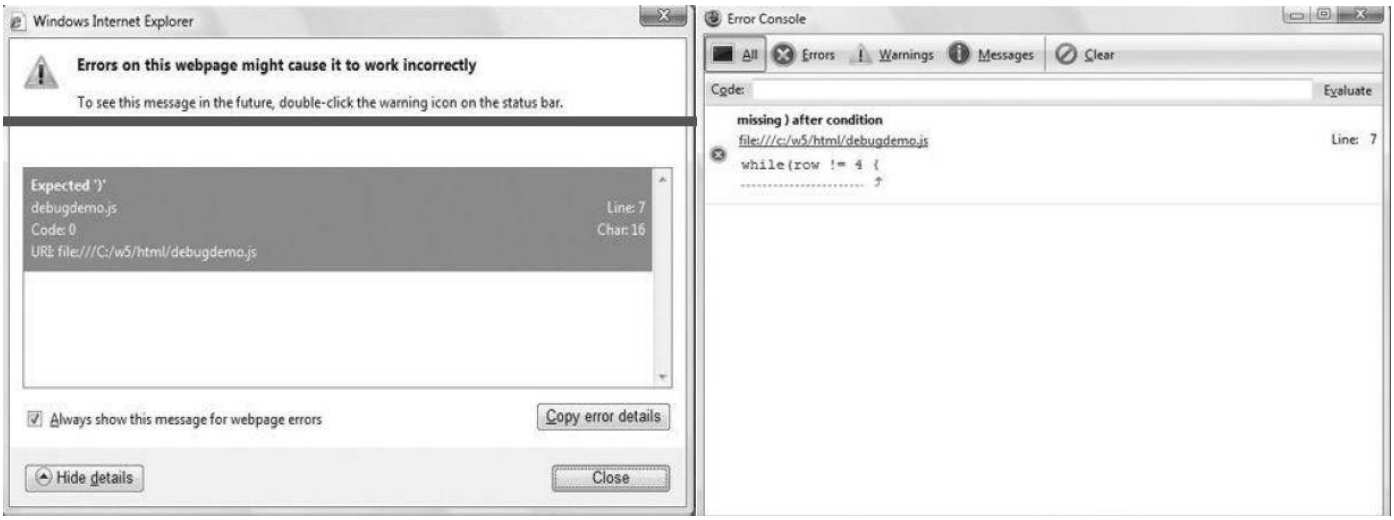
The JavaScript interpreter is capable of detecting various **errors in scripts**. Debugging a script is a bit different from debugging a program in a more typical programming language,

Because errors that are detected by the JavaScript interpreter are found while the browser is attempting to display a document.

In some cases, a script error causes the browser not to display the document and does not produce an error message. Without a diagnostic message, you must simply examine the code to find the problem.

```
// debugdemo.js
//    An example to illustrate debugging help

var row;
row = 0;
while(row != 4 {
   document.write("row is ", row, "<br />");
   row++;
}
```



Event and Event handling:

Event is an object, which is highly associated with the action of the mouse cursor on the web page. Such as ,

- A mouse click on an object in a web page.
- The movement of the mouse cursor across the web page.
- The mouse cursor hovering at a specific place on a web page and so on.

These will be the events recognized by the window object of the DOM.(document object model).Javascript's approach to the working with web page elements is a multi step process:

- Identify a web page object.
- Choose an appropriate even associated with the object.
- Have a standard method of connecting an object's event and javascript code snippets. Javascript **event handlers** mapped to an objects' events do this.
- Javascipt events are matched with appropriate event handlers.

In javascript there exist two types of event handlers. They are **Interactive and non-interactive**.

1. An interactive event handler depends on user interaction with an HTML page.

for example, onMouseOver event handler is interactive as it requires user to move mouse cursor on the web page.

2. Non-interactive event handler does not require user interaction to be invoked.

For example: the onLoad event handler is non-interactive event handler as it automatically executes whenever a form is loaded into a web page.

- In javascript event handler's name, simply has the string 'on' added to the HTML object's event name. (e.g: onMouseOver)

The following table shows event handlers that descriptively bound to HTML object events. As long as the HTML objects has as associated event, javascript provides an associated, named, event handler.

Named javascript event handler:

JavaScript Event Handler	Will Be Called When
onAbort	The loading of an image is aborted as a result of user action
onBlur	A document, window, frame set, or form element loses current input focus.
onChange	A text field, text area, file-uploaded field or selection is modified and loses the current input focus.
onClick	A link, client-side image map area or document is clicked
onDblClick	A link, client-side image map area or document is double clicked
onDragDrop	A dragged object is dropped in a window or frame
onError	An error occurs during loading of an image, window or frame
onFocus	A document, window, frame set, or form element receives the current input focus
onKeyDown	The user presses a Key
onKeyPress	The user presses and releases a Key
onKeyUp	The user releases a Key
onLoad	An image, document or frame set is loaded
onMouseDown	The user presses a mouse button
onMouseMove	The user moves the mouse
onMouseOut	The mouse is moved out of a link or an area of a client side image map
onMouseOver	The mouse is moved over a link or an area of a client side image map
onMouseUp	The user releases a mouse button
onReset	The user resets a form by clicking on the form's reset button
onResize	The user resizes a window or frame
onSelect	Text is selected in a text field or a text area
onSubmit	The user presses a form's submit button
onUnload	The user exits a document or frame set

Table 9.4

(*)