Dayananda Sagar University Bengaluru

Full Stack Development

Presentation Material

Department of Computer Science & Engineering

Course Code:	20CS2304	Semester:	3
Course Title:	FSD	Year:	II

MODULE 3

Overview of JavaScript; Object orientation and JavaScript; General syntactic, characteristics; Primitives, operations, and expressions; Screen output and keyboard input. Control statements; Arrays; Functions, Constructors; A brief introduction on pattern matching using regular expressions, DOM Events

Online

Resources:

https://www.javatpoint.com/javascript-tutorial

JavaScript

- JavaScript, which was originally developed at Netscape by Brendan Eich, was initially named Mocha but soon after was renamed LiveScript.
- In late 1995 LiveScript became a joint venture of Netscape and Sun Microsystems, and its name again was changed, this time to JavaScript.

Difference between Java and Javascript

- Java is a strongly typed language
- Variables in JavaScript need not be declared and are dynamically typed,2 making compile-time type checking impossible.
- objects in Java are static in the sense that their collection of data members and methods is fixed at compile time.
- JavaScript objects are dynamic: The number of data members and methods of an object can change during execution

Uses of JavaScript

- The original goal of JavaScript was to provide programming capability at both the server and the client ends of a Web connection.
- Document Object Model (DOM), which allows JavaScript scripts to access and modify the style properties and content of the elements of a displayed HTML document, making formally static documents highly dynamic.

Browsers and HTML-JavaScript Documents

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.
Disadvantages

1. Mixing two completely different kinds of notation in the same document makes the document difficult to read.
2. Second, in some cases, the person who creates and maintains the HTML is distinct from the person who creates and maintains the JavaScript

The JavaScript can be placed in its own file, separate from the HTML document. This approach, called implicit embedding.

<u>Object Orientation and JavaScript</u>

- JavaScript is not an object-oriented programming language.
- It is an object-based language. JavaScript does not have classes. Its objects serve both as objects and as models of objects.
- *Prototype-based inheritance*

<u>**JavaScript Objects**</u>

- 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 or a function or method property.
- Data properties appear in two categories: primitive values and references to other objects.
- JavaScript uses nonobject types for some of its simplest types; these nonobject types are called primitives.
- All objects in a JavaScript program are indirectly accessed through variables.

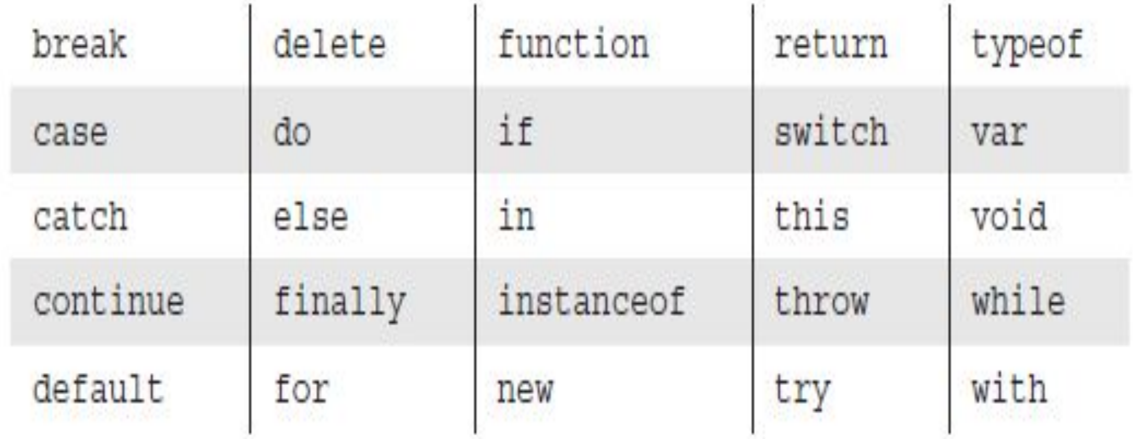
General Syntactic Characteristics

- 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 HTML document with the src attribute of a <script> tag, whose value is the name of a file that contains the script

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

- In JavaScript, identifiers, or names, are similar to those of other common programming languages.
- They must begin with a letter, an underscore (_), or a dollarsign ($).
- Subsequent characters may be letters, underscores, dollar signs, or digits.

Table 4.1 JavaScript reserved words



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

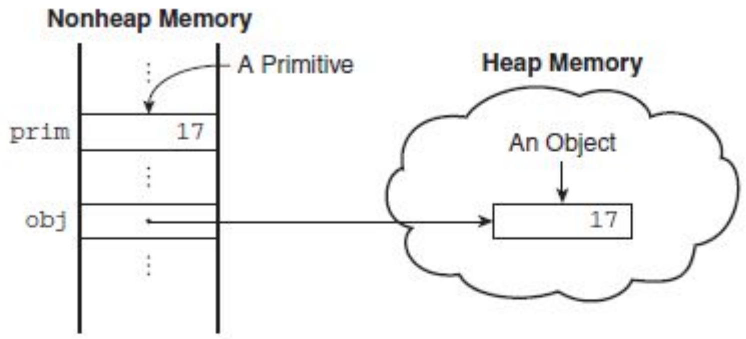
JavaScript has two forms of comments

1. Two adjacent slashes (//) single line comment
2. Multiple line comment /* may be used to introduce a comment, and */ to terminate it, in both single- and multiple-line comments.

```
<!DOCTYPE.html>
<!-- hello.html
A trivial hello world example of HTML/JavaScript
-->
<html lang = "en">
<head>
<title> Hello world </title>
<meta charset = "utf-8" />
</head>
<body>
<script type = "text/javascript">
<!--
document.write("Hello, fellow Web programmers!");
// -->
</script>
</body>
</html>
```

*Primitive, operations, and expressions

- JavaScript has five primitive types: Number, String, Boolean, Undefined, and Null
- *wrapper objects*



In JavaScript, objects are king. If you understand objects, you understand JavaScript.

In JavaScript, almost "everything" is an object.

- Boolean can be objects (if defined with the `new` keyword)
- Number can be objects (if defined with the `new` keyword)
- String can be objects (if defined with the `new` keyword)
- Date are always objects
- Math are always objects
- Regular expressions are always objects
- Array are always objects
- Function are always objects
- Object are always objects

* Numeric and String Literals

- All numeric literals are primitive values of type Number.
- The Number type values are represented internally in double-precision floating-point form.
- The following are valid numeric literals:

  72 7.2 .72 72. 7E2 7e2 .7e2 7.e2 7.2E-2

- A string literal is a sequence of zero or more characters delimited by either
- single quotes (') or double quotes (").
- String literals can include characters specified with escape sequences, such as \n and \t.
- Single-quote character in a string literal that is delimited by single quotes, the embedded single quote must be preceded by a backslash:

  'You\'re the most freckly person I\'ve ever seen‘

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

  "D:\\bookfiles"

Other Primitive Types

- 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.
- If an attempt is made to use the value of a variable whose value is null, it will cause a runtime error.
- The only value of type **Undefined** is undefined. Unlike null, there is no reserved word undefined.
- The only values of type **Boolean** are **true** and **false**

Declaring Variables

- A variable can be used for anything.
- Variables are **not typed;** values are.
- A variable can have the value of any primitive type, or it can be a reference to any object.
- The **type of the value** of a particular appearance of a variable in a program can be
- determined by the **interpreter.**
- **A variable can be declared** either by **assigning it a value**, 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***

Example

```
var counter;

pi = 3.14159265;
quarterback = "Elway";
stop_flag = true;
```

Numeric Operators

The binary operators

- + for addition, - for subtraction, * for multiplication, / for division, and %
- for modulus.
- The unary operators are plus (+), negate (-), decrement (--), and increment (++). The increment and decrement operators can be either prefix or postfix.

a=7

(++a) * 3=?

(a++) * 3=?

- All numeric operations are done in double-precision floating point.

- The *precedence rules* of a language specify which operator is evaluated first when two operators with different precedences 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

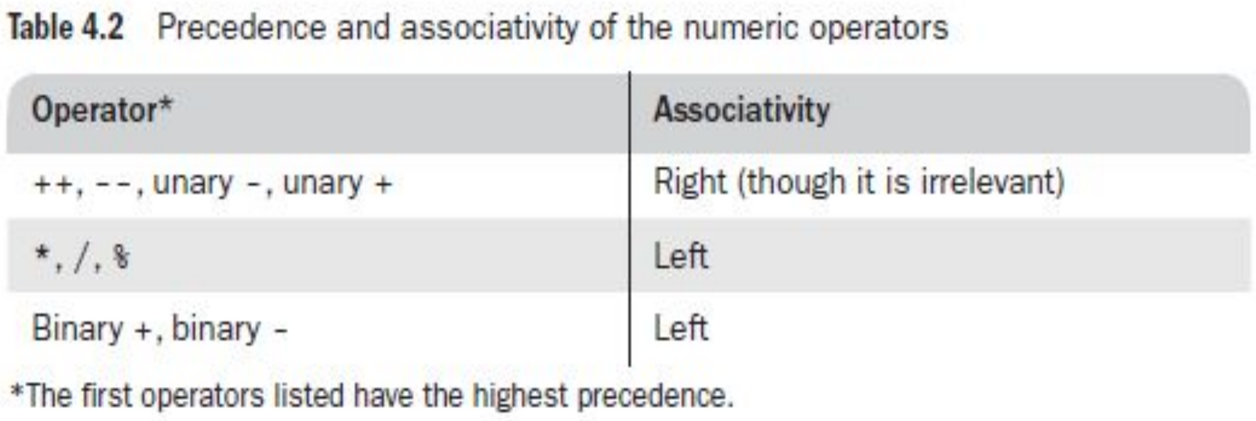


Table 4.2 Precedence and associativity of the numeric operators

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

*The first operators listed have the highest precedence.

```
var a = 2,
b = 4,


c = 3 + a * b; c=11
d = b / a / 2;
d=1
```

Parentheses can be used to force any desired precedence. For example, the
addition will be done before the multiplication in the following expression:

- (a + b) * c

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

<h2>My First JavaScript</h2>
<script>

var a=56;
b=87;

c=3*(a+b);
document.writeln(c);

</script>



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

</body>
</html>
```

In JavaScript, objects are king. If you understand objects, you understand JavaScript.

Built In Object: Math,Number,String,Date

Built in objects -Math Object

The Math Object(https://www.w3schools.com/js/js_math.asp)

- The Math object provides a collection of properties of Number objects and methods that operate on Number objects
- The Math object has methods for the trigonometric functions, such as **sin** and **cos**
- **floor,** to truncate a number;
- **round,** to round a number;
- **and max,** to return the largest of two given numbers.

https://www.w3schools.com/js/js_math.asp

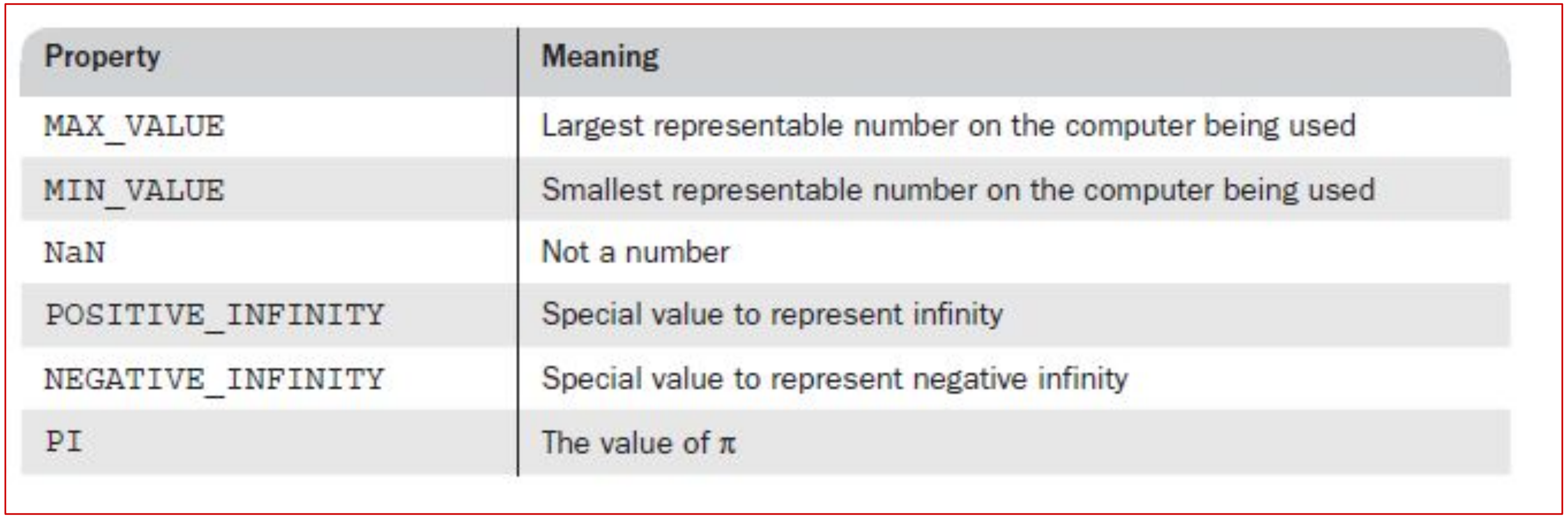
```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Math.round()</h2>

<p>Math.round(x) returns the value of x rounded to its nearest integer:</p>

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

<script>
document.getElementById("demo").innerHTML = Math.round(4.4);
</script>

</body>
</html>
```

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

<h2>JavaScript Math.sin()</h2>

<p>Math.sin(x) returns the sin of x (given in radians):</p>
<p>Angle in radians = (angle in degrees) * PI / 180.</p>

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

<script>
document.getElementById("demo").innerHTML =
"The sine value of 90 degrees is " + Math.sin(90 * Math.PI / 180);
</script>

</body>
</html>
```

Built in Object-The Number Object

- The **Number object** includes a collection of useful **properties** that have constant values.
- The **properties** are referenced through **Number.**

Example:Number.MIN_VALUE,Number.MAX_VALUE



Property	Meaning
MAX_VALUE	Largest representable number on the computer being used
MIN_VALUE	Smallest representable number on the computer being used
NaN	Not a number
POSITIVE_INFINITY	Special value to represent infinity
NEGATIVE_INFINITY	Special value to represent negative infinity
PI	The value of π

METHODS OF NUMBER OBJECT

Methods	Description
isFinite()	It determines whether the given value is a finite number.
isInteger()	It determines whether the given value is an integer.
parseFloat()	It converts the given string into a floating point number.
parseInt()	It converts the given string into an integer number.
toExponential()	It returns the string that represents exponential notation of the given number.
toFixed()	It returns the string that represents a number with exact digits after a decimal point.
toPrecision()	It returns the string representing a number of specified precision.
toString()	It returns the given number in the form of string.

refer https://www.javatpoint.com/javascript-number

```
<!DOCTYPE html>
<html>
<body>
<script>
var x=102;//integer value
var y=102.7;//floating point value
var z=13e4;//exponent value, output: 130000
var n=new Number(16);//integer value by number object
document.write(x+" "+y+" "+z+" "+n);
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<script>
var x=102;//integer value
var y=102.7;//floating point value
var z=13e4;//exponent value, output: 130000
var n=new Number(16);//integer value by number object
document.write(x+" "+y+" "+z+" "+n);
document.write(Number.MAX_VALUE);//max value possible in the computer
//document.write(n.MAX_VALUE);//cant call on instance since n=16,no max _value
document.write(Math.PI);
document.write(sqrt(4);)
document.write(x+y+z+n);

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

- The **Number object** has a method, **toString,** which it inherits from **Object** but overrides.
- The **toString method** converts the number through which it is called to a string.

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

<u>The String Catenation Operator</u>

- 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":
- first **+** " Freeloader"

Implicit Type Conversions

- The JavaScript interpreter performs several different implicit type conversions. Such conversions are called ***coercions.***
- For example,

  "August " + 1977

- The left operand is a string, the operator is considered to be a catenation operator. This forces string context on the right operand, so the right operand is implicitly converted to a string.

  "August 1997“

Explicit Type Conversions

var str_value = String(value);

This conversion can also be done with the toString method

- var num = 6;
- var str_value = num.toString();
- var str_value_binary = num.toString(2);

- **Strings can be explicitly converted to numbers in several different ways.**
- One way is with the **Number function**

  var number = Number(aString);

  eg

```
const quantity = "12";

console.log(typeof   quantity);// will return string


//We can convert quantity into a number using the Number function like this:


Number(quantity);
```

- The second way , **parseInt and parseFloat, are not String methods,** so they are not called through String objects. They operate on the strings given as parameters
- The parseInt function 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.**
- **The parseFloat function** searches for a floating-point literal, which could have a decimal point, an exponent, or both.

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



<script>
var a1=Number.parseInt("10") ; //or parseInt("10")

 var b1=parseInt("10.33");


var n1=a1+b1;  // adds 10+10, n1 will have 20


document.writeln ( n1);

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

```
<!DOCTYPE html>
<html>
<body>
<script>
var x=102;//integer value
var y=102.7;//floating point value
var z=13e4;//exponent value, output: 130000
var n=new Number(16);
document.write(x+" "+y+" "+z+" "+n);
document.write("<br>");
document.write(x+y+z+n);
document.write("<br>");
document.writeln(Number.parseInt("10"));
document.writeln("<br>");
x=Number.parseInt("20");
y=Number.parseInt("40");
z=x+y;
   l=parseFloat("40.2");
document.writeln("the value of z is"+ "<br>"+ z
+"<br>");
document.writeln("the value of l is"+ "<br>"+
l+"<br>");
x=Number.parseInt("40 50 12");
document.writeln("the value of x is"+ "<br>"+ x+
"<br>");
t=parseInt("hello 12");
document.writeln("the value of l is"+ "<br>"+
t+"<br>");
d=parseInt(" 12 hello ");
document.writeln("the value of d is"+ "<br>"+ d+
"<br>");
document.writeln(x.toExponential());converts
number to string in exponent form
document.writeln(x.toString());//converts number
to string
</script>
</body>
</html>
```

```
<!DOCTYPE html>     // EXAMPLE REPEATED WITH EVENTS AND FUNCTION
<html>
<body>

<p>Click the button to parse different strings.</p>

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

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

<p id="demo1"></p>
<script>
function myFunction() {
var a=Number.parseInt("10");
//  var a = parseInt("10") + "<br>";

  var b = parseInt("10.00") + "<br>";
  var c = parseInt("10.33") + "<br>";
  var d = parseInt("34 45 66") + "<br>";//parse Int takes only the first argument and converts it into integer.
  var e = parseInt("   60   ") + "<br>";
  var f = parseInt("40 years") + "<br>";
  var g = parseInt("He was 40") + "<br>";

  var h = parseInt("10", 10)+ "<br>";
  var i = parseInt("010")+ "<br>";
  var j = parseInt("10", 8)+ "<br>";
  var k = parseInt("0x10")+ "<br>";
  var l = parseInt("10", 16)+ "<br>";

  var n = a + b + c + d + e + f + g + "<br>" + h + i + j + k +l;                 //no addition here ,only we are displaying the converted value
  document.getElementById("demo").innerHTML = n;
 var a1=Number.parseInt("10") ; //or parseInt("10")

 var b1=parseInt("10.33");


var n1=a1+b1;  // adds 10+10, n1 will have 20

document.getElementById("demo1").innerHTML = n1;


}
</script>

</body>
</html>
```

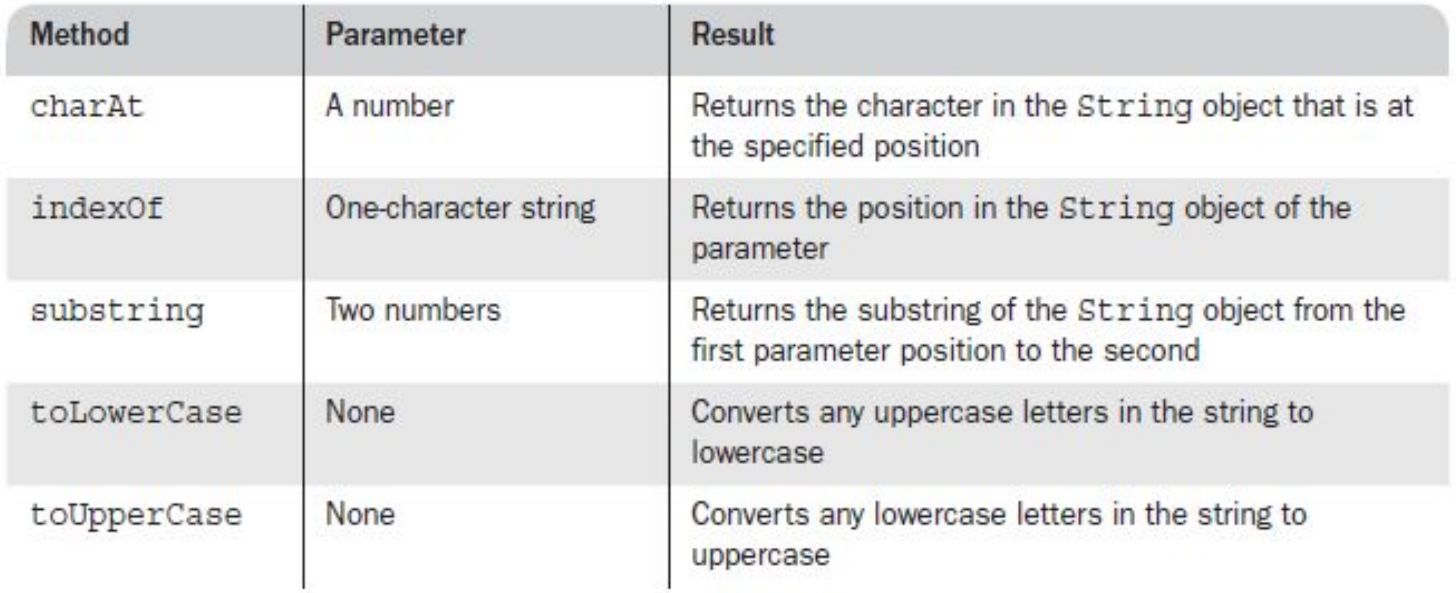
Built in Object-**String** Properties and Methods

- String methods can always be used through String primitive values, as if the values were objects.
- The String object includes one property, length, and a large collection of methods.
- The number of characters in a string is stored in the length property

```
var str = "George";
var len = str.length;
```

len is set to the number of characters in str, namely, 6.

*String Methods



Method	Parameter	Result
`charAt`	A number	Returns the character in the `String` object that is at the specified position
`indexOf`	One-character string	Returns the position in the `String` object of the parameter
`substring`	Two numbers	Returns the substring of the `String` object from the first parameter position to the second
`toLowerCase`	None	Converts any uppercase letters in the string to lowercase
`toUpperCase`	None	Converts any lowercase letters in the string to uppercase

- Example

var str = "George";

Then the following expressions have the values shown:

- str.charAt(2) will display 'o'
- str.indexOf('r') will display 3
- str.substring(2, 5) will display 'org'
- str.toLowerCase() will display 'george'

display like- document.writeln(str.charAt(2));
https://www.javatpoint.com/javascript-string

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.

- If the operand is an object or null, typeof **produces "object“**
- If the operand is a variable that has not been assigned a value, **typeof** produces **"undefined“**

Assignment Statements

- There is a simple assignment operator, denoted by =, and a host of compound assignment operators, such as += and /=.

- a += 7;

means the same as

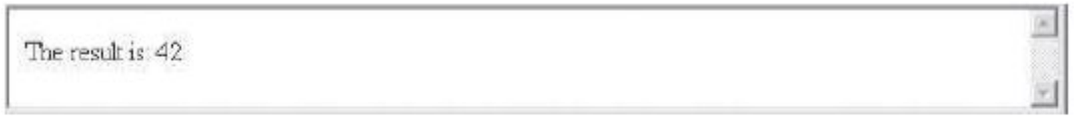
- a = a + 7;

```
<html>
<body>

<h1>JavaScript Operators</h1>
<h2>The typeof Operator</h2>

<p>The typeof operator returns the type of a variable, object,
function or expression:</p>

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

<script>
document.getElementById("demo").innerHTML =
"'John' is " + typeof "John" + "<br>" +
"3.14 is " + typeof 3.14 + "<br>" +
"NaN is " + typeof NaN + "<br>" +
"false is " + typeof false + "<br>" +
"[1, 2, 3, 4] is " + typeof [1, 2, 3, 4] + "<br>" +
"{name:'John', age:34} is " + typeof {name:'John', age:34} +
"<br>" +
"new Date() is " + typeof new Date() + "<br>" +
"function () {} is " + typeof function () {} + "<br>" +
"myCar is " + typeof myCar + "<br>" +
"null is " + typeof null;
</script>

</body>
</html>
```

*Screen Output and Keyboard Input

- JavaScript models the HTML document with the Document object.
- The window in which the browser displays an HTML document is modeled with the Window object.
- The Window object includes two properties, document and window.
- The document property refers to the Document object.
- The window property is self-referential; it refers to the Window object.
- The Document object has several properties and methods. Commonly used is **write,** which is used to create output, which is dynamically created HTML document content.

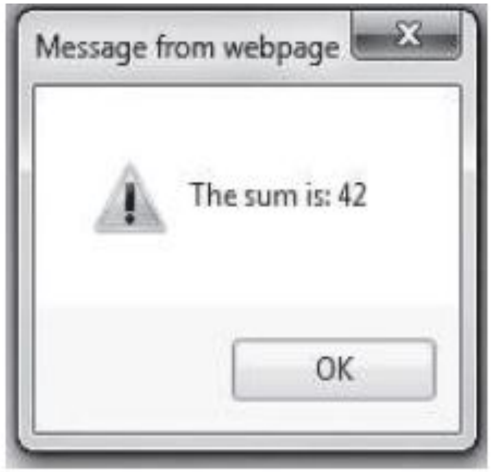
```
document.write("The result is: ", result, "<br />");
```



```
The result is: 42
```

- The parameter of **write** can include any HTML tags and content.
- The write method actually can take any number
- of parameters. Multiple parameters are catenated and placed in the output.
- **Window includes three methods that create dialog boxes for three specific kinds of user interactions.**
- **The three methods— *alert, confirm, and prompt***
- The alert method opens a dialog window and displays its parameter in
  that window. It also displays an *OK* button.

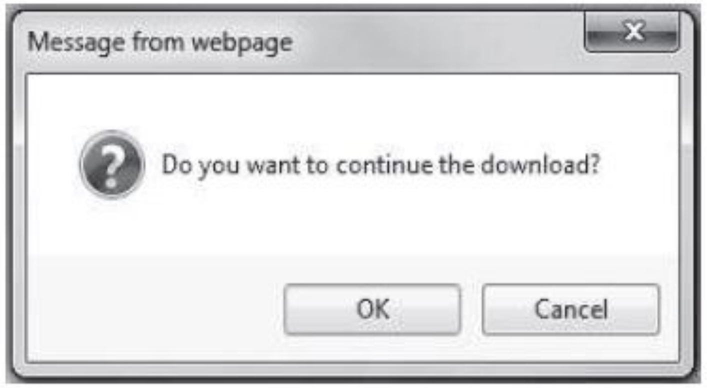
```
alert("The sum is:" + sum + "\n");
```


Message from webpage

The sum is: 42

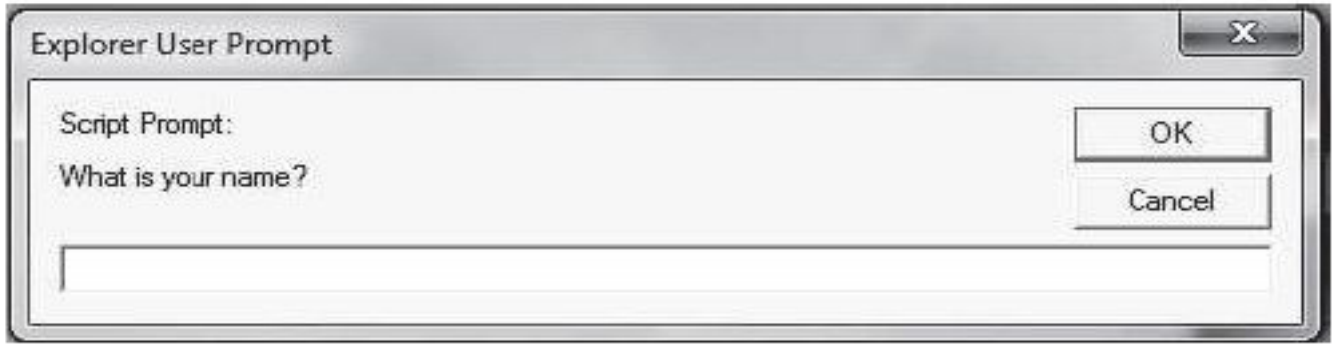
- **The confirm method opens a dialog window in which the method displays**
- **its string parameter, along with two buttons: *OK* and *Cancel*.**
- confirm returns a Boolean value that indicates the user's button input: true for *OK* and false for *Cancel*.

- var question = confirm("Do you want to continue this download?");


Message from webpage

Do you want to continue the download?

- **The prompt method** creates a dialog window that contains a text box used
  to collect a string of input from the user, which prompt returns as its value.
- Prompt window includes two buttons: ***OK* and *Cancel*.**
- Prompt takes two parameters: the string that prompts the user for input and a default string in case the user does not type a string before pressing one of the two buttons.
- name = prompt("What is your name?", "");



```
<!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

```
var a = prompt("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", "");
var root_part = Math.sqrt(b * b - 4.0 * a * c);
var denom = 2.0 * a;
// Compute and display the two roots
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 />");
```

Control Statements

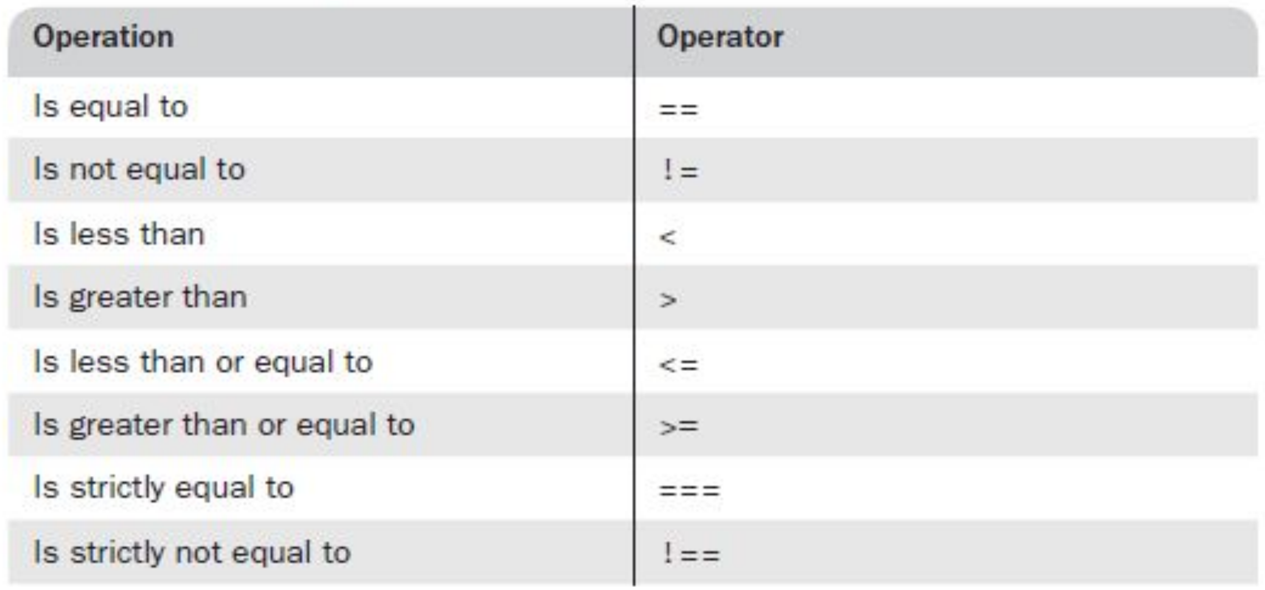
- Control statements often require some syntactic container for sequences of statements whose execution they are meant to control.
- In JavaScript, that container is the compound statement.
- A *compound statement* in JavaScript is a sequence of statements delimited by braces.
- A *control construct* is a control statement together with the statement or compound statement whose execution it controls.

Control Expressions

- The expressions upon which statement flow control can be based include primitive values, relational expressions, and compound expressions.
- The result of evaluating a control expression is one of the Boolean values true or false

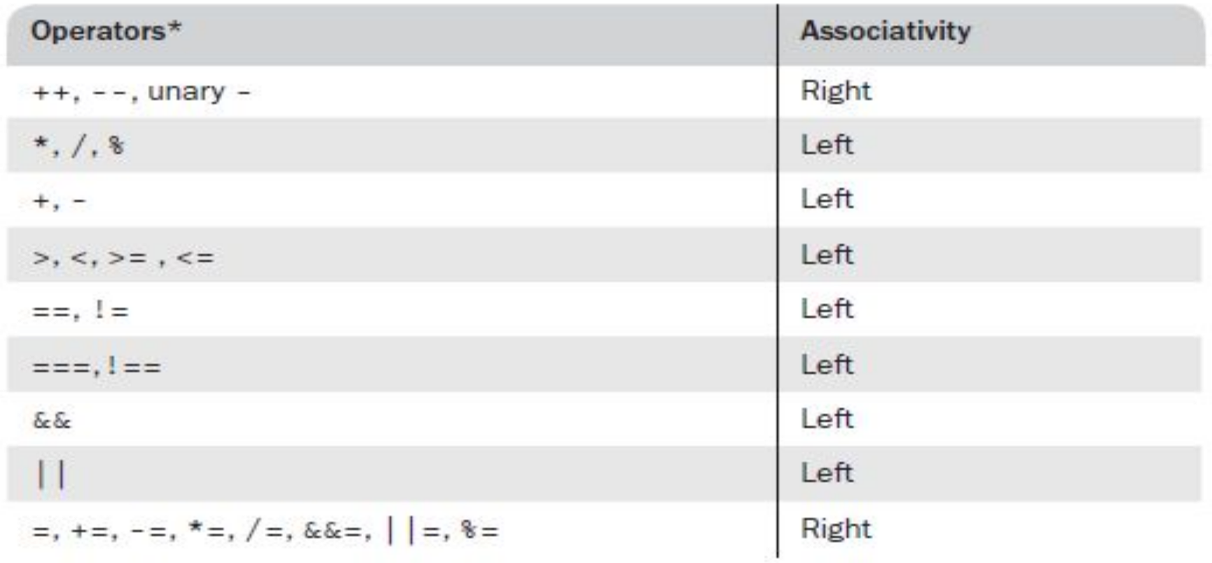
- A relational expression has two operands and one relational operator.

Table 4.6 Relational operators



Operation	Operator
Is equal to	==
Is not equal to	!=
Is less than	<
Is greater than	>
Is less than or equal to	<=
Is greater than or equal to	>=
Is strictly equal to	===
Is strictly not equal to	!==

- If the two operands in a relational expression are not of the same type and the operator is neither === nor !==, JavaScript will attempt to convert the operands to a single type.
- If a and b reference different objects, a == b is never true, even if the objects have identical properties. a == b is true only if a and b reference the same object.



Operators*	Associativity
++, --, unary -	Right
*, /, %	Left
+, -	Left
>, <, >= , <=	Left
==, !=	Left
===, !==	Left
&&	Left
||	Left
=, +=, -=, *=, /=, &&=, ||=, %=	Right

Selection Statements

- The selection statements (if-then and if-then-else). Either single statements or compound statements can be selected

```
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

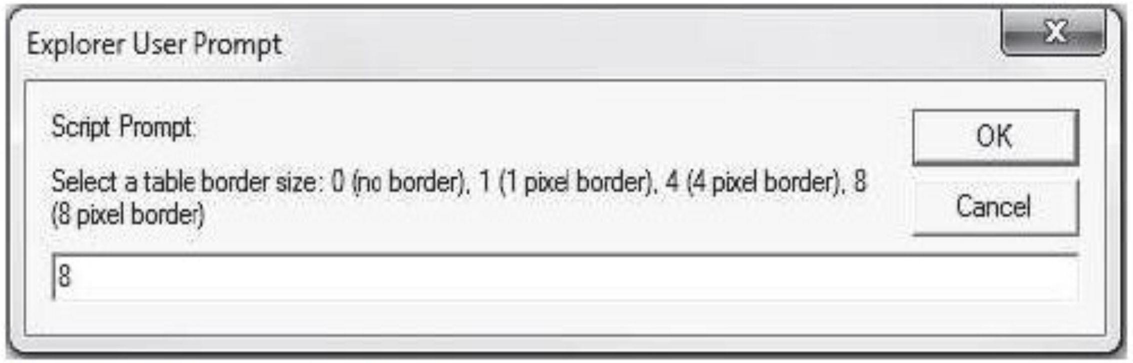
```
switch (expression) {
case value_1:

// statement(s)
case value_2:
// statement(s)
...
[default:
// statement(s) ]
}
```

Note :Save the next program as borders2.js

write HML code to call border2.js

```
var bordersize;
var err = 0;
bordersize = prompt("Select a table border size: " +
"0 (no border), " +
"1 (1 pixel border), " +
"4 (4 pixel border), " +
"8 (8 pixel border), ");
switch (bordersize) {
case "0": document.write("<table>");
break;
case "1": document.write("<table border = '1'>");
break;
case "4": document.write("<table border = '4'>");
break;
case "8": document.write("<table border = '8'>");
break;
default: {
document.write("Error - invalid choice: ",
bordersize, "<br />");
err = 1;
}
}
If (err == 0) {
document.write("<caption> 2012 NFL Divisional",
" Winners </caption>");
```

```
document.write("<tr>",
"<th />",
"<th> American Conference </th>",
"<th> National Conference </th>",
"</tr>",
"<tr>",
"<th> East </th>",
"<td> New England Patriots </td>",
"<td> Washington Redskins </td>",
"</tr>",
"<tr>",
"<th> North </th>",
"<td> Baltimore Ravens </td>",
"<td> Green Bay Packers </td>",
"</tr>",
"<tr>",
"<th> West </th>",
"<td> Denver Broncos </td>",
"<td> San Francisco 49ers </td>",
"</tr>",
"<tr>",
"<th> South </th>",
"<td> Houston Texans </td>",
"<td> Atlanta Falcons </td>",
"</tr>",
"</table>");}
```



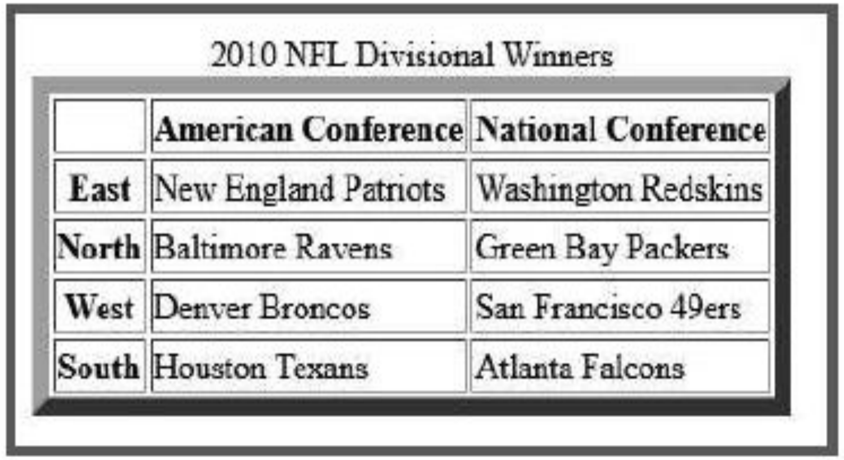
Explorer User Prompt

Script Prompt:

Select a table border size: 0 (no border), 1 (1 pixel border), 4 (4 pixel border), 8 (8 pixel border)

8

Figure 4.6 Dialog box from `borders2.js`



2010 NFL Divisional Winners

	American Conference	National Conference
East	New England Patriots	Washington Redskins
North	Baltimore Ravens	Green Bay Packers
West	Denver Broncos	San Francisco 49ers
South	Houston Texans	Atlanta Falcons

Figure 4.7 Display produced by `borders2.js`

Loop Statements

while (*control expression*)
statement or compound statement

The general form of the for statement is as follows:

for (*initial expression*; *control expression*; *increment expression*) *statement or compound statement*

- Both the initial expression and the increment expression of the for statement can be multiple expressions separated by commas

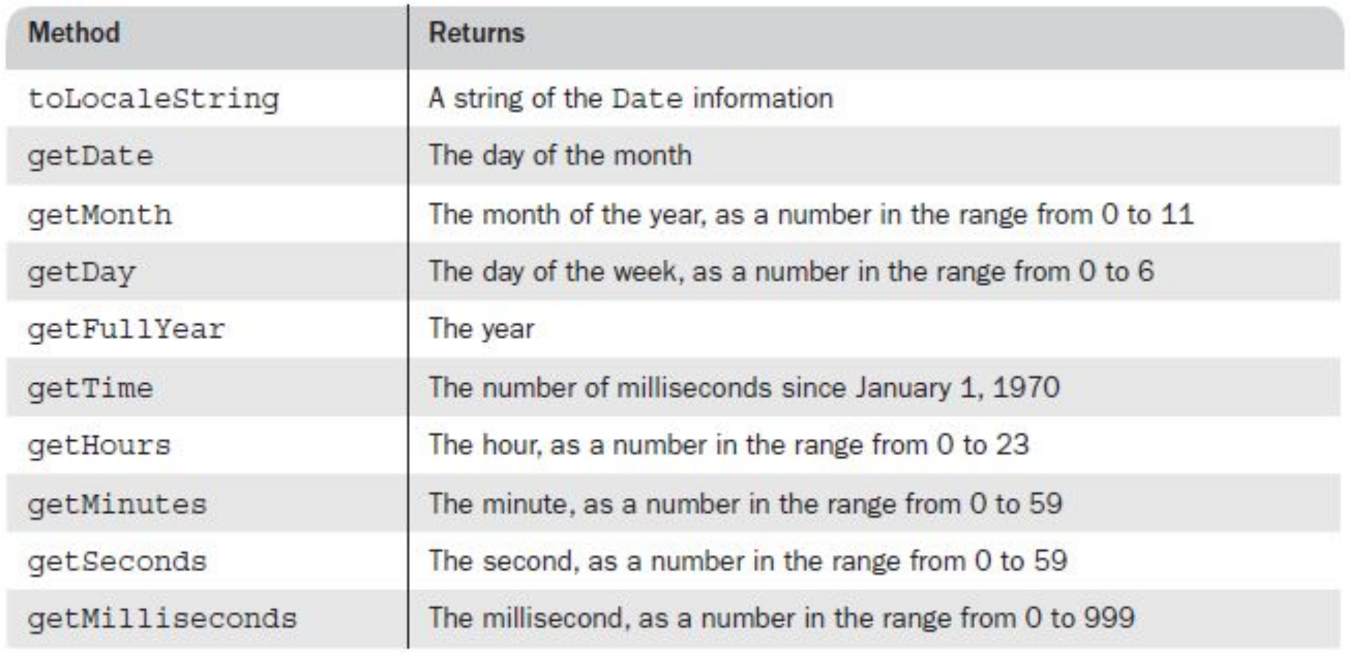
```
    var sum = 0,   count;
for (count = 0; count <= 10; count++)
 sum += count;
```

*Date Object

The **Date** Object

tutorial reference: https://www.javatpoint.com/javascript-date

- A Date object is created with the new operator and the Date constructor, which has several forms.
- var today = new Date();
- The date and time properties of a Date object are in two forms: local and Coordinated Universal Time (UTC)



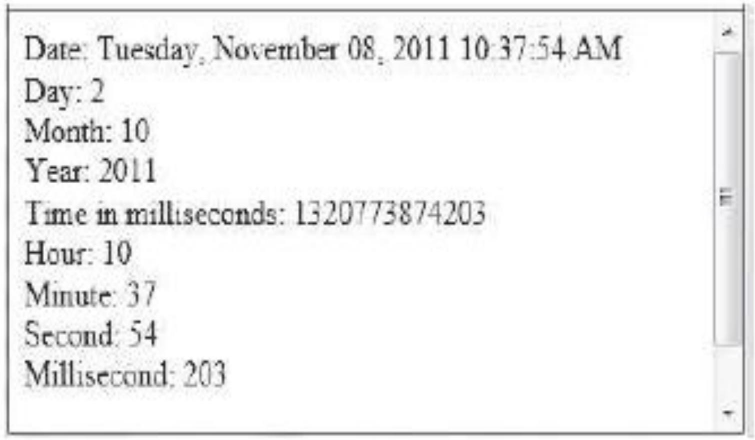
Method	Returns
`toLocaleString`	A string of the `Date` information
`getDate`	The day of the month
`getMonth`	The month of the year, as a number in the range from 0 to 11
`getDay`	The day of the week, as a number in the range from 0 to 6
`getFullYear`	The year
`getTime`	The number of milliseconds since January 1, 1970
`getHours`	The hour, as a number in the range from 0 to 23
`getMinutes`	The minute, as a number in the range from 0 to 59
`getSeconds`	The second, as a number in the range from 0 to 59
`getMilliseconds`	The millisecond, as a number in the range from 0 to 999

```
// Get the current date
var today = new Date();
// Fetch the various parts of the date
var dateString = today.toLocaleString();
var day = today.getDay();
var month = today.getMonth();
var year = today.getFullYear();
var timeMilliseconds = today.getTime();
var hour = today.getHours();
var minute = today.getMinutes();
var second = today.getSeconds();
var millisecond = today.getMilliseconds();
// Display the parts
document.write(
"Date: " + dateString + "<br />",
"Day: " + day + "<br />",
"Month: " + month + "<br />",
"Year: " + year + "<br />",
"Time in milliseconds: " + timeMilliseconds + "<br />",
"Hour: " + hour + "<br />",
"Minute: " + minute + "<br />",
"Second: " + second + "<br />",
"Millisecond: " + millisecond + "<br />");
// Time a loop
var dum1 = 1.00149265, product = 1;
var start = new Date();
for (var count = 0; count < 10000; count++)
product = product + 1.000002 * dum1 /
1.00001;
var end = new Date();
var diff = end.getTime() - start.getTime();
document.write("<br />The loop took " + diff +
" milliseconds <br />");
```

// Note: date.js

save the program as date.js

call this date.js in html program.



```
Date: Tuesday, November 08, 2011 10:37:54 AM
Day: 2
Month: 10
Year: 2011
Time in milliseconds: 1320773874203
Hour: 10
Minute: 37
Second: 54
Millisecond: 203
```

Figure 4.8 Display produced by `date.js`

dowhile

do *statement or compound statement*
while (*control expression*)

- The body of a do-while construct is always executed at least once

```
do {
count++;
sum = sum + (sum * count);
} while (count <= 50);
```

Objects

https://www.w3schools.com/js/js_objects.asp
https://www.sitepoint.com/back-to-basics-javascript-object-syntax/
https://www.javascript.com/learn/objects

Window

Window is the main JavaScript object root, aka the global object in a browser, and it can also be treated as the root of the document object model. You can access it as window.

window.screen or just **screen** is a small information object about physical screen dimensions.

window.document or **just document** is the **main object of the potentially visible (or better yet: rendered) document** object model/DOM.

Since window is the global object, *you can reference any properties of it with just the property name - so you do not have to write down window. - it will be figured out by the runtime.*

window.document just means that document is a **property** of window. It's not an instance of window. window is the *global* object. Every global variable is a property of the global object

The Document Object

When an HTML document is loaded into a web browser, it becomes a document object.

The document object is the root node of the HTML document.

The document object is a property of the window object.

The document object is accessed with:

`window.document` **or just** `document`

Object Creation and Modification

- 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, **the new operator** creates a blank object—that is, one with no properties.
- **JavaScript objects do not have types. T**he constructor both creates and initializes
- the properties.

The following statement creates an object that has no properties:

```
var my_object = new Object();
var my_object={};
```

- **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.
- **Properties are not actually variables**—they are just the names of values. They are used with object variables to access property values.
- properties are not variables, they are never declared.
- **The number of properties in a JavaScript object is** dynamic. 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 its name.**

```
// Create an Object object
var my_car = new Object();
// Create and initialize the make property
my_car.make = "Ford";
// Create and initialize model
my_car.model = "Fusion";
```

- creates a new object, my_car, with two properties: make and model

The object referenced with my_car could be created with the following statement:

- var my_car = {make: "Ford", model: "Fusion"};

Objects can be nested, you can create a new object that is a property of my_car with properties of its own, as in the following statements:

- my_car.engine = new Object();
- my_car.engine.config = "V6";
- my_car.engine.hp = 263;

- **Properties can be accessed in two ways.**
- First, any property can be accessed in the same way it is assigned a value, namely, with the object-dot-property notation.
- Second, the property names of an object can be accessed as if they were elements of an array.

```
var prop1 = my_car.make;
var prop2 = my_car["make"];
```

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

If an attempt is made to access a property of an object that does not exist, the value undefined is used.
A property can be deleted with delete, as in the following example:

- delete my_car.model;

- JavaScript has a loop statement, **for-in, t**hat is perfect for listing the properties
  of an object.
  for (identifier in object)
  statement or compound statement

Example:

```
for (var prop in my_car)
document.write("Name: ", prop, "; Value: ", my_car[prop], "<br />");
```

program on object//we are creating a custom object called car

```
<html>
<body>
<p id="demo"></p>
<script>

var car= new Object();
car.type="maruthi";
car.speed="500"
car.engine=new Object();
car.engine.type="diesel";          //nested property
car.engine.xxx="llll";

document.getElementById("demo").innerHTML = car["speed"];
document.getElementById("demo").innerHTML = car.engine.type;
delete car.engine.xxx;
for(var i in car)     //note new for loop
{
document.writeln("property is ", i, ";its value is",car[i],"<br>");
}



</script>

</body>
</html>
```

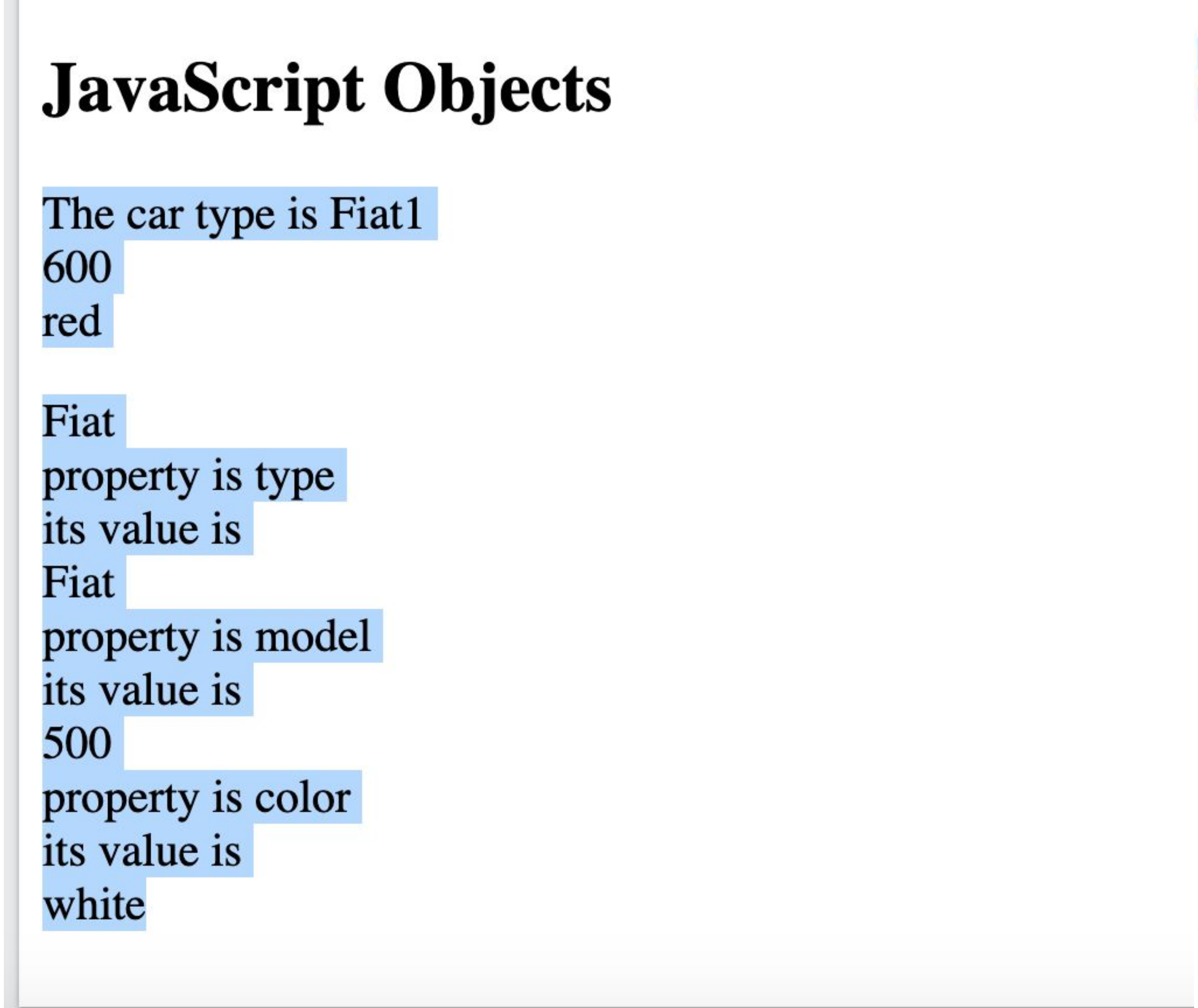
//or for loop can have contcatenation sign

```
{
document.writeln("property is "+i+" its value is"+car[i],"<br>");
document.writeln("<br>");
}
```

Output

```
diesel

property is type;its value ismaruthi
property is speed;its value is500
property is engine;its value is[object Object]
```

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Objects</h2>
<p id="demo"></p>
<p id="demo1"></p>
<script>
// Create an object in two ways
const car = {type:"Fiat",
model:"500", color:"white"};
var car1= new Object();
car1.type="Fiat1";
car1.model="600";
car1.color="red";
// Display some data from the object:
document.getElementById("demo").innerHTML =
"The car type is " + car1.type+"<br>"+ car1.model
+"<br>"+ car1.color

document.writeln(car["type"]);// fiat
for(var i in car)
{
document.writeln("<br> property is "+i+ "<br> its
value is <br>"+car[i]);
}



</script>

</body>
</html>
```



JavaScript Objects

```
The car type is Fiat1
600
red

Fiat
property is type
its value is
Fiat
property is model
its value is
500
property is color
its value is
white
```

Dayananda Sagar University Bengaluru

*Object Example

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

<h2>JavaScript Objects</h2>

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

<script>
// Create an object:
var car = {type:"Fiat", model:"500", color:"white"};   // this is alternate way of creating car object

// Display some data from the object:
document.getElementById("demo").innerHTML = "The car type is " + car.type;
</script>

</body>
</html>
```

output: Fiat

Object example

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

<h2>JavaScript Objects</h2>

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

<script>
// Create an object:
var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

*ARRAYS

EXAMPLE ON ARRAYS WITH ITS PROPERTIES

EXAMPLE ON ARRAYS WITH ITS METHODS

*Arrays javascript.com/learn/arrays

- Array objects, unlike most other JavaScript objects, can be created in two distinct ways.
- The usual way to create any object is to apply the new operator to a call to a constructor.

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

- The second way to create an Array object is with a literal array value, which

is a list of values enclosed in brackets:

```
var my_list_2 = [1, 2, "three", "four"];
```

Characteristics of Array Objects

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

```
my_list[47] = 2222;
```

- 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.
- The length of an array can be set to whatever you like by assigning the **length property,** as in the following example:
- **my_list.length** = 1002;
- To support dynamic arrays of JavaScript, all array elements are allocated dynamically from the heap.

Extra program to practice

```
// insert_names.js
//   The script in this document has an array of
//   names, name_list, whose values are in
//   alphabetic order. New names are input
through
//   prompt. Each new name is inserted into the
//   name array, after which the new list is
//   displayed.
```

Extra problem on -Insert names into the sorted list

```
!DOCTYPE html>

<!-- insert_names.html
     A document for insert_names.js
     -->
<html lang = "en">
  <head>
    <title> Name list </title>
    <meta charset = "utf-8" />
  </head>
  <body>
    <script type = "text/javascript"  src = "insert_names.js" >
    </script>
  </body>
</html>
```

```
// insert_names.js
//   The script in this document has an array of
//   names, name_list, whose values are in
//   alphabetic order. New names are input through
//   prompt. Each new name is inserted into the
//   name array, after which the new list is
//   displayed.
// The original list of names

   var name_list = new Array("Al", "Betty", "Kasper",

            "Michael", "Roberto", "Zimbo");
   var new_name, index, last;

// Loop to get a new name and insert it

   while (new_name =
          prompt("Please type a new name", "")) {

// Loop to find the place for the new name

    last = name_list.length - 1;

    while (last >= 0 && name_list[last] > new_name) {
      name_list[last + 1] = name_list[last];
      last--;
    }
// Insert the new name into its spot in the array

        name_list[last + 1] = new_name;

// Display the new array

        document.write("<p><b>The new name list is:</b> ",
                       "<br />");
        for (index = 0; index < name_list.length; index++)
          document.write(name_list[index], "<br />");

/* There is another way to go over every element as below:
       for(a in name_list)
         document.write(name_list[a], "<br />");
*/

        document.write("</p>");
      } //** end of the outer while loop
```

*Array Methods

https://www.w3schools.com/js/js_array_methods.asp

Array Methods

- **The join method converts all 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.

var names = new Array("Mary", "Murray", "Murphy", "Max");
var name_string = names.join(" : ");
//try working out, : will join all the array contents with :
//ans is :join() returns an array as string //u can try joining with any characters like 'and'

Mary:Murray:Murphy:Max

- **reverse method:** It reverses the order of the elements of the Array object through which it is called. Eg **var reversedstring =names.reverse();**
- **The sort method** coerces the elements of the array to become strings if they

are not already strings and sorts them alphabetically.
For example: names.sort();
The value of names is now "Mary", "Max", "Murphy", "Murray".

- The concat method catenates its actual parameters to the end of the Array object on which it is called.

var names = new Array["Mary", "Murray", "Murphy", "Max"];
...
var new_names = names.concat("Moo", "Meow");

- **The new_names array** now has length 6, with the elements of names, along with "Moo" and "Meow" as its fifth and sixth elements.

- **The slice method:** returning the part of the Array object specified by its parameters, which are used as subscripts.

var list = [2, 4, 6, 8, 10];
var list2 = list.slice(1, 3);

- The value of list2 is now [4, 6].

- If slice is given just one parameter, the array that is returned has all the elements of the object, starting with the specified index.
- var list = ["Bill", "Will", "Jill", "dill"];
- **var listette = list.slice(2);**

- the value of listette is set to ["Jill", "dill"].

- When the **toString method is called through an Array object,** each of the elements of the o**bject is converted to a string.**
- These strings are catenated, separated by commas. So, for Array objects, the toString method behaves much like join.

- **The push, pop, unshift, and shift methods of Array allow the easy implementation of stacks and queues in arrays.**
- **The pop and push methods** remove and add an element to the high end of an array, var list = ["Dasher", "Dancer", "Donner", "Blitzen"];

```
var deer = list.pop(); // deer is now "Blitzen"
list.push("Blitzen");
```

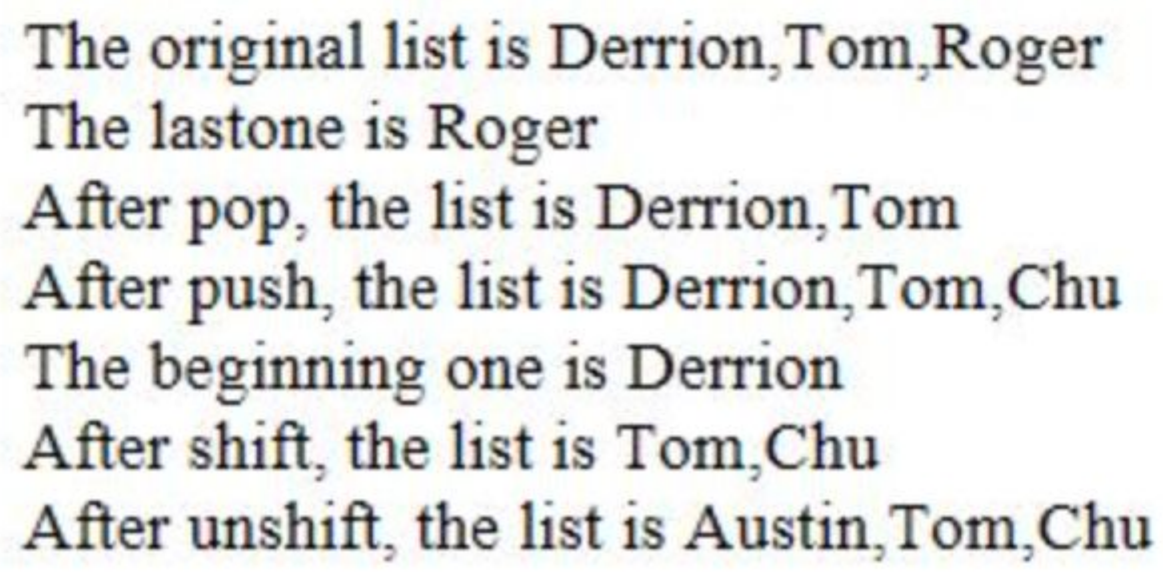
- **The shift and unshift methods** remove and add an element to the beginning of an array.

```
var deer = list.shift();
list.unshift("Dasher");
```

- 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

Example

```
var list = new Array("Derrion", "Tom", "Roger");
document.write("The original list is ", list, "<br />");
var lastone = list.pop();
document.write("The lastone is ", lastone, "<br />");
document.write("After pop, the list is ", list, "<br />");
list.push("Chu");
document.write("After push, the list is ", list, "<br />");
var beginning = list.shift();
document.write("The beginning one is ", beginning, "<br />");
document.write("After shift, the list is ", list, "<br />");
list.unshift("Austin");
document.write("After unshift, the list is ", list, "<br />");
```

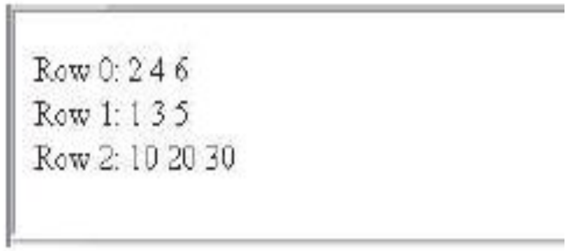


```
The original list is Derrion,Tom,Roger
The lastone is Roger
After pop, the list is Derrion,Tom
After push, the list is Derrion,Tom,Chu
The beginning one is Derrion
After shift, the list is Tom,Chu
After unshift, the list is Austin,Tom,Chu
```

Extra problem to practice

```
<!DOCTYPE html>

<!-- nested_arrays.html
     A document for nested_arrays.js
     -->
<html lang = "en">
  <head>
    <title> Array of arrays </title>
    <meta charset = "utf-8" />
  </head>
  <body>
    <script type = "text/javascript"  src = "nested_arrays.js" >
    </script>
  </body>
</html>
```

```
// 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 />");
    }
```



```
Row 0: 2 4 6
Row 1: 1 3 5
Row 2: 10 20 30
```

example program of array methods

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

<script>
var arr=["AngularJS","Node.js","JQuery","Bootstrap"]
var arr5=[1,3,6,7,2]
var arr6 =arr5.sort(function(a,b){return b-a});   //sorting in descending order
document.writeln("after sortin");
document.writeln(arr6);
var result=arr.slice(1,2);//note:changes are not done in the original array,so store in new array
var result1=arr.concat("graphql","react");
document.writeln(result);
//following also does sorting in descending order using bubble sort
var ar = new Array(3,1,5,6);

for( var i = 0 ; i < ar.length ; i++)
{
    for( var j = 0 ; j < ar.length-1;j++)
    {
        if ( ar[j] < ar[j+1])
        {
```

```
var temp = ar[j] ;
            ar[j] = ar[j+1] ;
            ar[j+1] = temp ;
        }

    }

}



document.write(ar);

document.writeln("<br>");
document.writeln(result1);
document.writeln(arr);
```

```
rev1=arr.reverse();
rev2=arr5.reverse();
document.writeln("<br>");
document.writeln(rev2);
document.writeln("i am using
push,pop,shift,unshift ,changes done in
original array");
arr.push("graphql","jsx");
//arr.concat("graphql","react");
arr.shift();
arr.unshift("css");
//document.writeln(s);
document.writeln(arr);
</script></body></html>
```

Functions

Functions

- A *function definition* consists of the function’s header and a compound statement that describes the actions of the function.
- This compound statement is called the *body* of the function.
- A function *header* consists of the reserved word function, the function’s name, and a parenthesized list of parameters if there are any
- A return statement returns control from the function in which it appears to the function’s caller.
- fun1();
- result = fun2();
- 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.

- function fun() { document.write(
- "This surely is fun! <br/>");}
- ref_fun = fun; // Now, ref_fun refers to the fun object
- fun(); // A call to fun
- ref_fun(); // Also a call to fun

Local Variables

- The *scope* of a variable is the range of statements over which it is visible. When
- JavaScript is embedded in an HTML document, the scope of a variable is the range of lines of the document over which the variable is visible.
- Variables that are implicitly declared have *global scope*—that is, they are visible in the entire HTML document.
- Variables that are
- Explicitly declared outside function definitions also have global scope. As stated earlier, we recommend that all variables be explicitly declared.

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*.

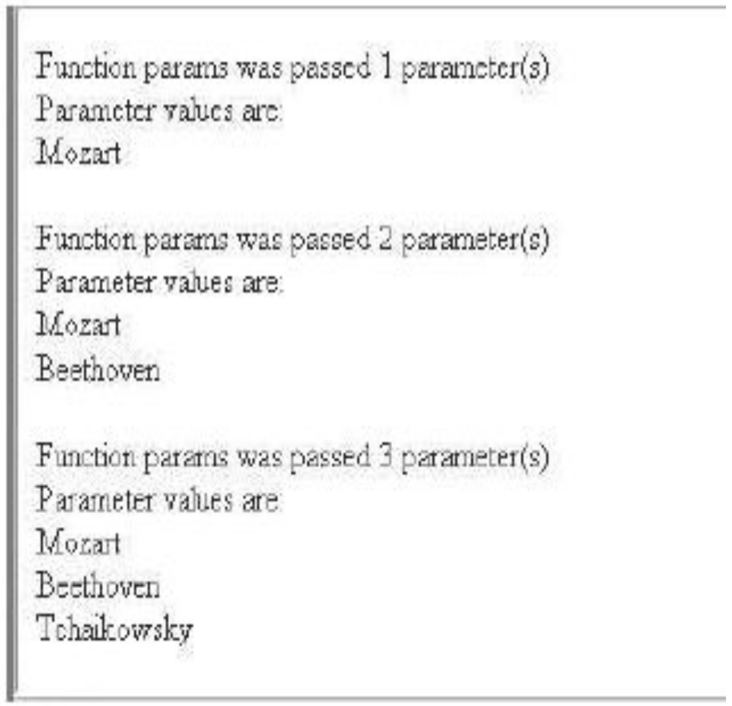
```
function fun1(my_list) {
var list2 = new Array(1, 3, 5);
my_list[3] = 14;
...
my_list = list2;
}
...
var list = new Array(2, 4, 6, 8)
fun1(list);
```

*FUNCTION CAN BE STORED AS VARIABLE

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

<h2>JavaScript Functions</h2>
<p>After a function has been stored in a
variable,
the variable can be used as a function:</p>

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

<script>
const x = function (a, b) {return a * b};
document.getElementById("demo").innerHTML
= x(4, 3);
</script>

</body>
</html>
```

Example: function in a java script

```
function params(a, b) {
document.write("Function params was passed ",
arguments.length, " parameter(s) <br />");
document.write("Parameter values are: <br />");
for (var arg = 0; arg < arguments.length; arg++)
document.write(arguments[arg], "<br />");
document.write("<br />");
}
// A test driver for function params
params("Mozart");
params("Mozart", "Beethoven");
params("Mozart", "Beethoven", "Tchaikowsky");
```



```
Function params was passed 1 parameter(s)
Parameter values are:
Mozart

Function params was passed 2 parameter(s)
Parameter values are:
Mozart
Beethoven

Function params was passed 3 parameter(s)
Parameter values are:
Mozart
Beethoven
Tchaikowsky
```

pass a primitive value by reference

- There is no elegant way in JavaScript to pass a primitive value by reference.
- One inelegant way is to put the value in an array and pass the array, as in the following script:

```
// Function by10
// Parameter: a number, passed as the first element
// of an array
// Returns: nothing
// Effect: multiplies the parameter by 10
function by10(a) {
a[0] *= 10;
}
...
var x;
var listx = new Array(1);
...
listx[0] = x;
by10(listx);
x = listx[0];
```

- Another way to have a function change the value of a primitive-type actual parameter is to have the function return the new value as follows:

```
function by10_2(a) {
return 10 * a;
}
var x;
...
x = by10_2(x);
```

The **sort** Method, Revisited

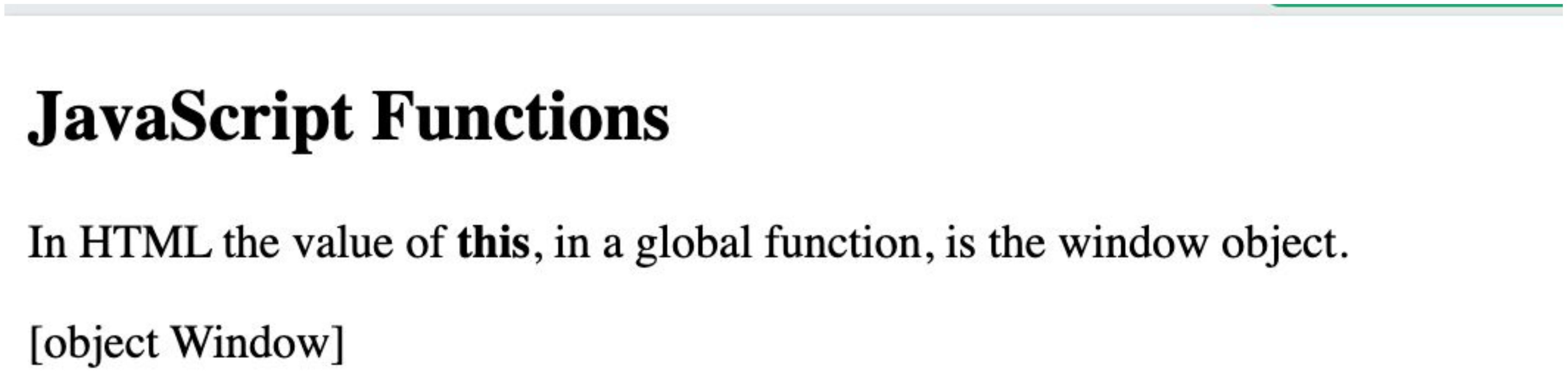
- If you need to sort something other than strings, or if you want an array to be sorted in some order other than alphabetically as strings, the comparison operation must
  be supplied to the sort method by the caller. Such a comparison operation is
passed as a parameter to sort.

Example:

```
num_list.sort(num_order);
```

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

<h2>JavaScript Functions</h2>

<p>Global functions automatically become window methods. Invoking myFunction() is the same as
invoking window.myFunction().</p>

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

<script>
function myFunction(a, b) {
  return a * b;
}
document.getElementById("demo").innerHTML = window.myFunction(10, 2);
</script>

</body>
</html>
```

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

<h2>JavaScript Functions</h2>

<p>In HTML the value of <b>this</b>, in a global function, is the window object.</p>

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

<script>
let x = myFunction();
function myFunction() {
  return this;
}
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```



JavaScript Functions

In HTML the value of **this**, in a global function, is the window object.

```
[object Window]
```

*Invoking functions as method

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

<h2>JavaScript Functions</h2>

<p>myObject.fullName() will return John Doe:</p>

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

<script>
const myObject =
{
  firstName:"John",
  lastName: "Doe",

  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}
document.getElementById("demo").innerHTML = myObject.fullName();
document.getElementById("demo").innerHTML=myObject.firstName;
</script>

</body>
</html>
```

The second method of creating a JavaScript object is to use the constructor function

. We define an object type without any specific values. Then, we create new object instances and populate each of them with different values.

Below, we can see the same *userProfile001* object defined by using a constructor function called *function User()*. The constructor creates an object type called *User()*. Then, we create a new object instance called *userProfile001*, using the *new operator*. The constructor function contains three *this* statements that define the three properties with empty values. The values of the properties are added by each object instance

Constructors

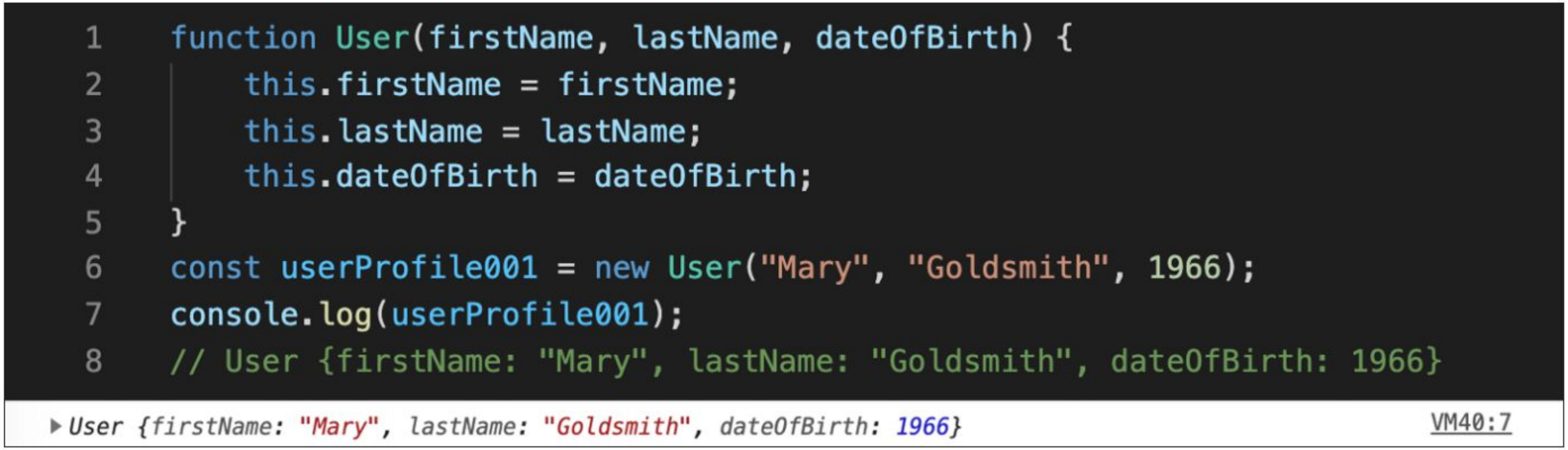
Dayananda Sagar University Bengaluru

Constructors

- JavaScript constructors are special functions that create and initialize the properties of newly created objects
- Every new expression must include a call to a constructor whose name is the same as that of the object being created.

```
<script>
function car(new_make, new_model, new_year)
{

this.make = new_make;
this.model = new_model;
this.year = new_year;

this.display=function display_car() {
document.write("Car make: ", this.make, "<br/>");
document.write("Car model: ", this.model, "<br/>");
document.write("Car year: ", this.year, "<br/>");
}
}


my_car = new car("Ford", "Fusion", "2012");

</script>
```

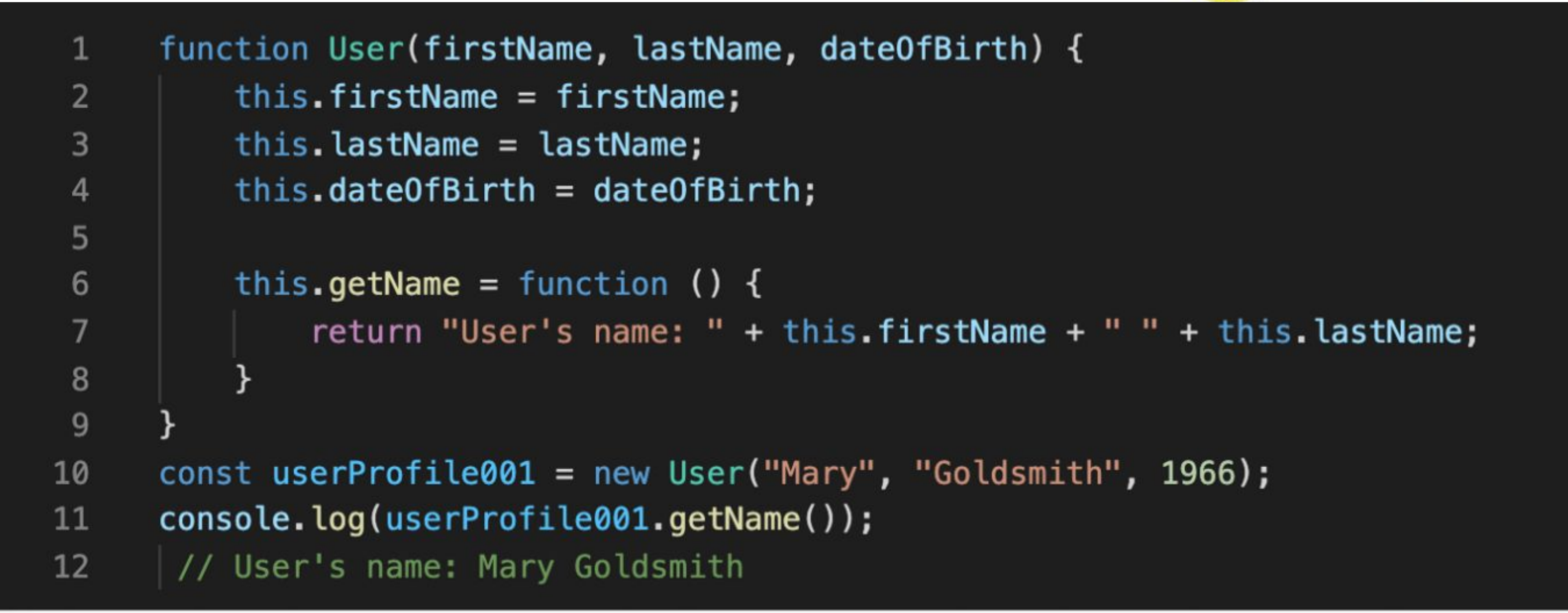
- If a method is to be included in the object, it is initialized the same way as if it were a data property example2



```
function User(firstName, lastName, dateOfBirth) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.dateOfBirth = dateOfBirth;
}
const userProfile001 = new User("Mary", "Goldsmith", 1966);
console.log(userProfile001);
// User {firstName: "Mary", lastName: "Goldsmith", dateOfBirth: 1966}
```

```
▶ User {firstName: "Mary", lastName: "Goldsmith", dateOfBirth: 1966}                VM40:7
```

example 3



```
function User(firstName, lastName, dateOfBirth) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.dateOfBirth = dateOfBirth;

    this.getName = function () {
        return "User's name: " + this.firstName + " " + this.lastName;
    }
}
const userProfile001 = new User("Mary", "Goldsmith", 1966);
console.log(userProfile001.getName());
// User's name: Mary Goldsmith
```

Example 4 Invoking a Function with a Function Constructor

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

<h2>JavaScript Functions</h2>
<p>In this example, myFunction is a function constructor:</p>

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

<script>
function myFunction(arg1, arg2) {
  this.firstName = arg1;
  this.lastName  = arg2;
return this;
}

const myObj = new myFunction("John","Doe")
document.getElementById("demo").innerHTML = myObj.firstName;
</script>

</body>
</html>
```

DEFAULT FUNCTION CONSTRUCTOR

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

<h2>JavaScript Functions</h2>
<p>JavaScript has an built-in function
constructor.</p>
<p id="demo"></p>

<script>
const myFunction = new Function("a", "b", "return a *
b");
document.getElementById("demo").innerHTML =
myFunction(4, 3);
</script>

</body>
</html>
```

ALTERNATIVELY

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

<h2>JavaScript Functions</h2>
<p id="demo"></p>

<script>
const myFunction = function (a, b)
{return a * b}
document.getElementById("demo").in
nerHTML = myFunction(4, 3);
</script>

</body>
</html>
```

more egs:

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

<h2>JavaScript Functions</h2>
<p>Finding the largest number.</p>
<p id="demo"></p>

<script>

function findMax() {
 let max = Number.NEGATIVE_INFINITY;
 document.writeln(Number.NEGATIVE_INFINITY);
  //let  max=-Infinity;
  for(let i = 0; i < arguments.length; i++) {
    if (arguments[i] > max) {
      max = arguments[i];
                                          }
                                                            }
  return max;
                                  }

document.getElementById("demo").innerHTML = findMax(4, 5, 6);
</script>

</body>
</html>
```

Arrow function

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

<h2>JavaScript Arrow Functions</h2>

<p>Arrow functions are not supported in IE11 or earlier.</p>

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

<script>
//document.getElementById("demo").innerHTML = x(5, 5);
const x = (x, y) => {return x * y };
document.getElementById("demo").innerHTML = x(5, 5);
</script>

</body>
</html>
```

The following line must then be added to the car constructor:

```
this.display = display_car;
```

Now the call my_car.display() will produce the following output:

```
Car make: Ford
Car model: Fusion
Car year: 2012
```

Extra program to practice-Table program using javascript

Dayananda Sagar University Bengaluru

```
<html>
<head>
<script>
document.write("<table border='1'><tr><th colspan='3'>"+ "NUMBERS FROM 0 TO 10 WITH THEIR
SQUARES AND CUBES" +"</th></tr>" );
document.write("<tr><th>Number</th><th>Square</th><th>Cube</th></tr>");
for(var n=1; n<=10; n++)
{
document.write( "<tr><td>" + n + "</td><td>" + n*n +
"</td><td>" +
n*n*n + "</td></tr>" ) ;
}
document.write( "</table>" );
</script>
</head>
</html>
```

Dayananda Sagar University Bengaluru

*DOM-WHAT IS DOM

* WHAT ARE EVENTS(DIFFERENT TYPES OF EVENTS

* REGISTERING AN EVENT

- JavaScript binding to the DOM, the elements of a document are objects, with
  both data and operations. The data are called *properties, and the operations are,*
  naturally, called *methods.*

  For example, the following :

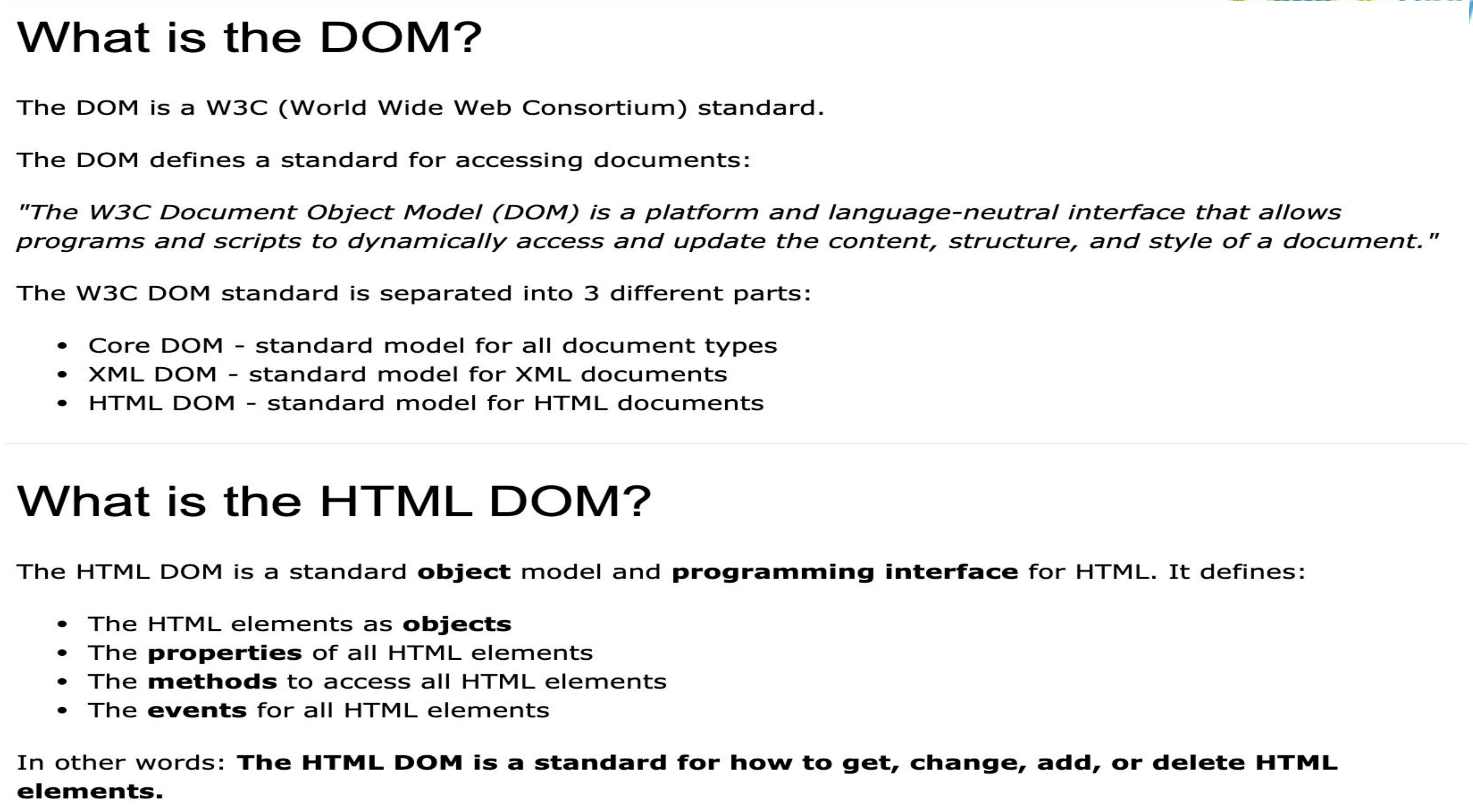
- *HTML element would be* represented as an object with two properties, type and name, with the values "text" and "address", respectively:

```
<input type = "text" name = "address">
```

- In most cases, the property names in JavaScript are the same as their corresponding attribute names in HTML.

Document Object Model :

- DOM is an Application Programming Interface (API) that defines an interface between HTML documents and application programs.
- Documents in the DOM have a treelike structure, but there can be more than one tree in a document (although that is unusual).
- Because the DOM is an abstract interface, it does not dictate that documents be implemented as trees or collections of trees.



What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

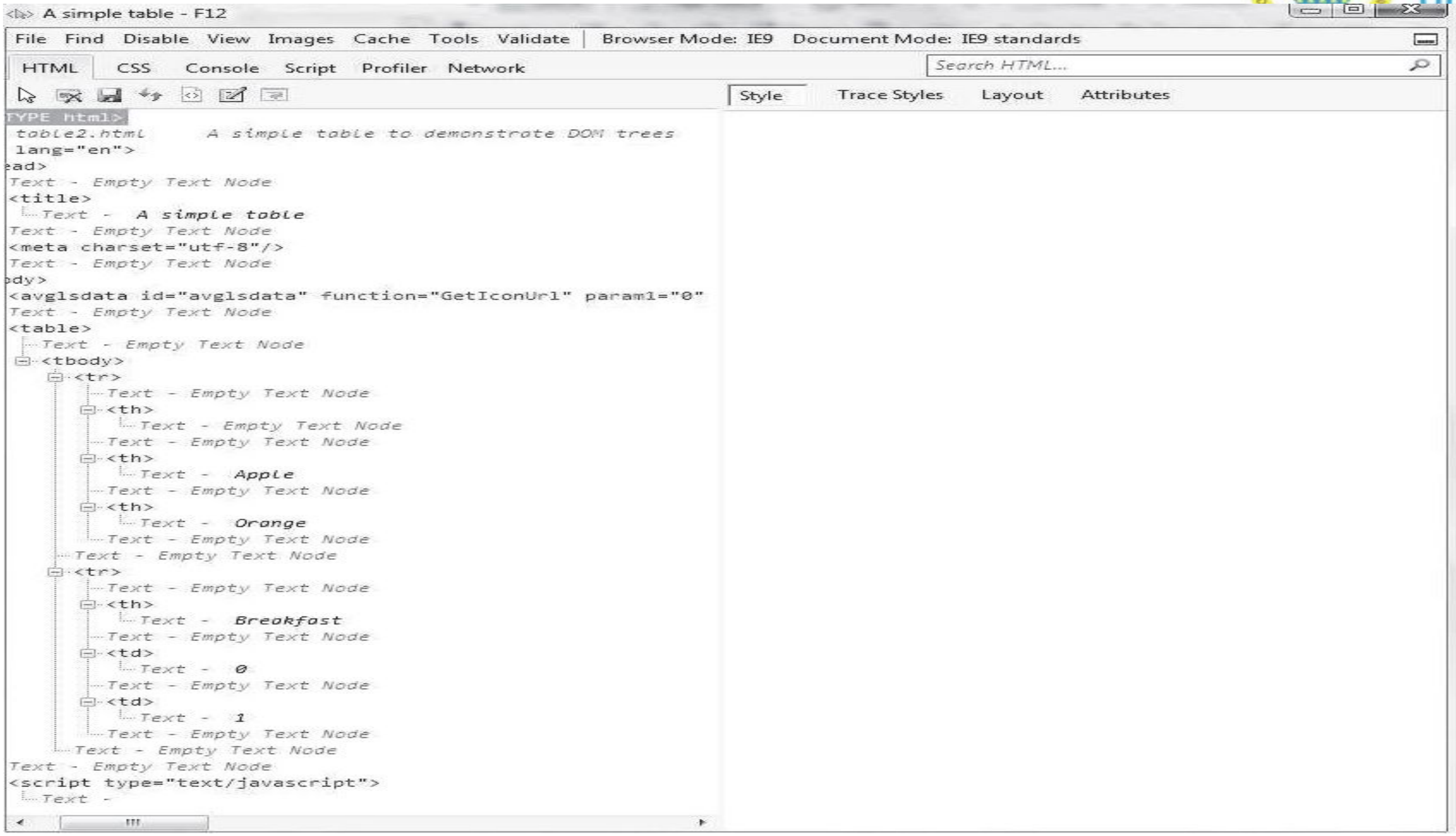
In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

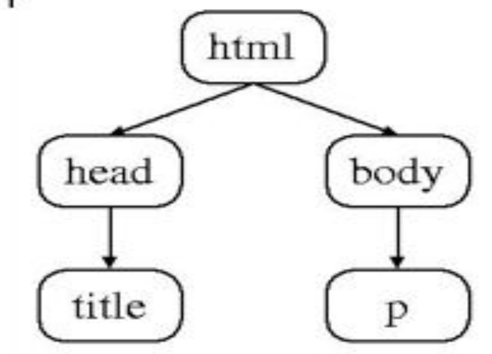
```html
<!DOCTYPE html>
<!-- table2.html
A simple table to demonstrate DOM trees
-->
<html lang = "en">
<head>
<title> A simple table </title>
<meta charset = "utf-8" />
</head>
<body>
<table>
<tr>
<th> </th>
<th> Apple </th>
<th> Orange </th>
</tr>
<tr>
<th> Breakfast </th>
<td> 0 </td>
<td> 1 </td>
</tr>
</table>
</body>
</html>
```


A simple table - F12
File Find Disable View Images Cache Tools Validate Browser Mode: IE9 Document Mode: IE9 standards
HTML CSS Console Script Profiler Network
Search HTML...
Style Trace Styles Layout Attributes

```
TYPE html>
 table2.html    A simple table to demonstrate DOM trees
 lang="en">
ead>
Text - Empty Text Node
<title>
  Text -  A simple table
Text - Empty Text Node
<meta charset="utf-8"/>
Text - Empty Text Node
dy>
<avglsdata id="avglsdata" function="GetIconUrl" param1="0"
Text - Empty Text Node
<table>
  Text - Empty Text Node
  <tbody>
    <tr>
      Text - Empty Text Node
      <th>
        Text - Empty Text Node
      Text - Empty Text Node
      <th>
        Text -  Apple
      Text - Empty Text Node
      <th>
        Text -  Orange
      Text - Empty Text Node
    Text - Empty Text Node
    <tr>
      Text - Empty Text Node
      <th>
        Text -  Breakfast
      Text - Empty Text Node
      <td>
        Text -  0
      Text - Empty Text Node
      <td>
        Text -  1
      Text - Empty Text Node
    Text - Empty Text Node
Text - Empty Text Node
<script type="text/javascript">
  Text -
```

Document Tree

- Recall that HTML document elements form a tree structure, *e.g.*,



- DOM allows scripts to access and modify the document tree

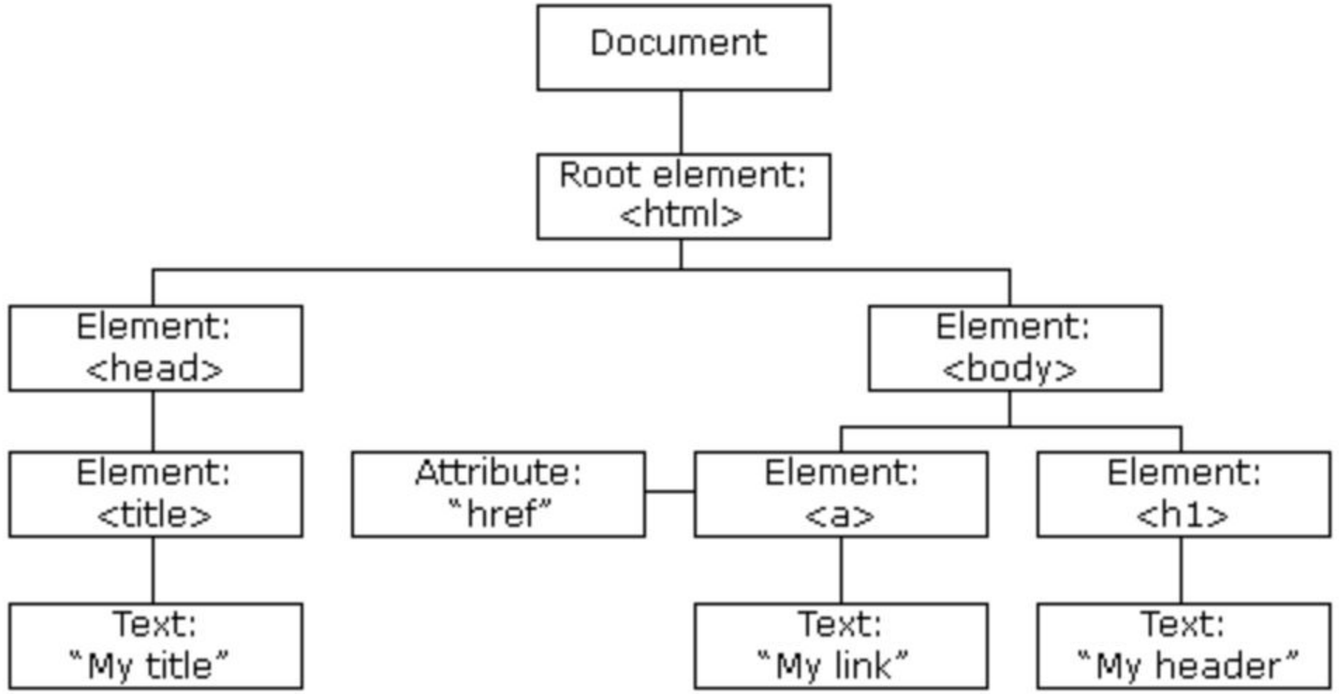
x

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

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

The HTML DOM Document Object

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method	**Description**
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Changing HTML Elements

Property	**Description**
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

Adding and Deleting Elements

Method	**Description**
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Events and Event Handling:

- Event-driven: code executed resulting to user or browser action.
- Event: a notification that something specific occurred -- by browser or user.
- Event handler: a script implicitly executed in response to event occurrence.
- Registration: **the process of connecting event handler to event.**

- **Events** are JavaScript objects --> names are **case sensitive**, all use lowercase only.
  (Method ***write*** should never be used in event handler. May cause document to be written over.)
- JavaScript **events** associated with HTML tag attributes which can be used to **connect to** event-handlers

- **JavaScript's interaction with HTML** is handled through **events** that occur when the user or the browser **manipulates** a page.
- When the **page loads**, it is called an **event**. When the **user clicks a button,** that click too is an **event**,Other examples include **events like pressing any key, closing a window, resizing a window, etc.**
- Developers can use these events to execute JavaScript coded **responses,** which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.
- Events are a part of the **Document Object Model DOM Level 3** and every

- HTML element contains a set of events which can trigger JavaScript Code.

Events, Attributes, and Tags:

- HTML4 defined a collection of events that browsers

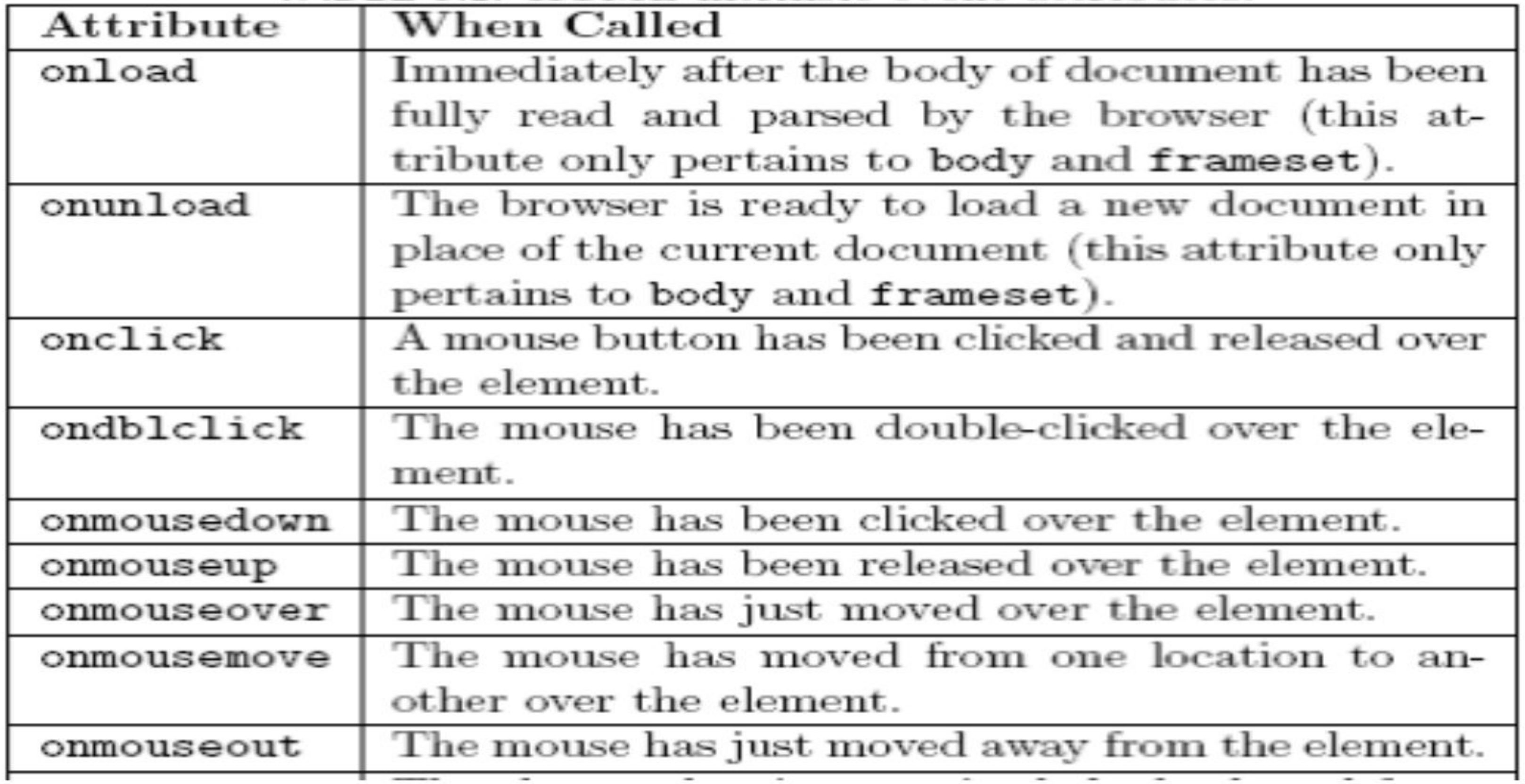
  implement and with which
- JavaScript can deal. **These events are associated with HTML**

  tag attributes,

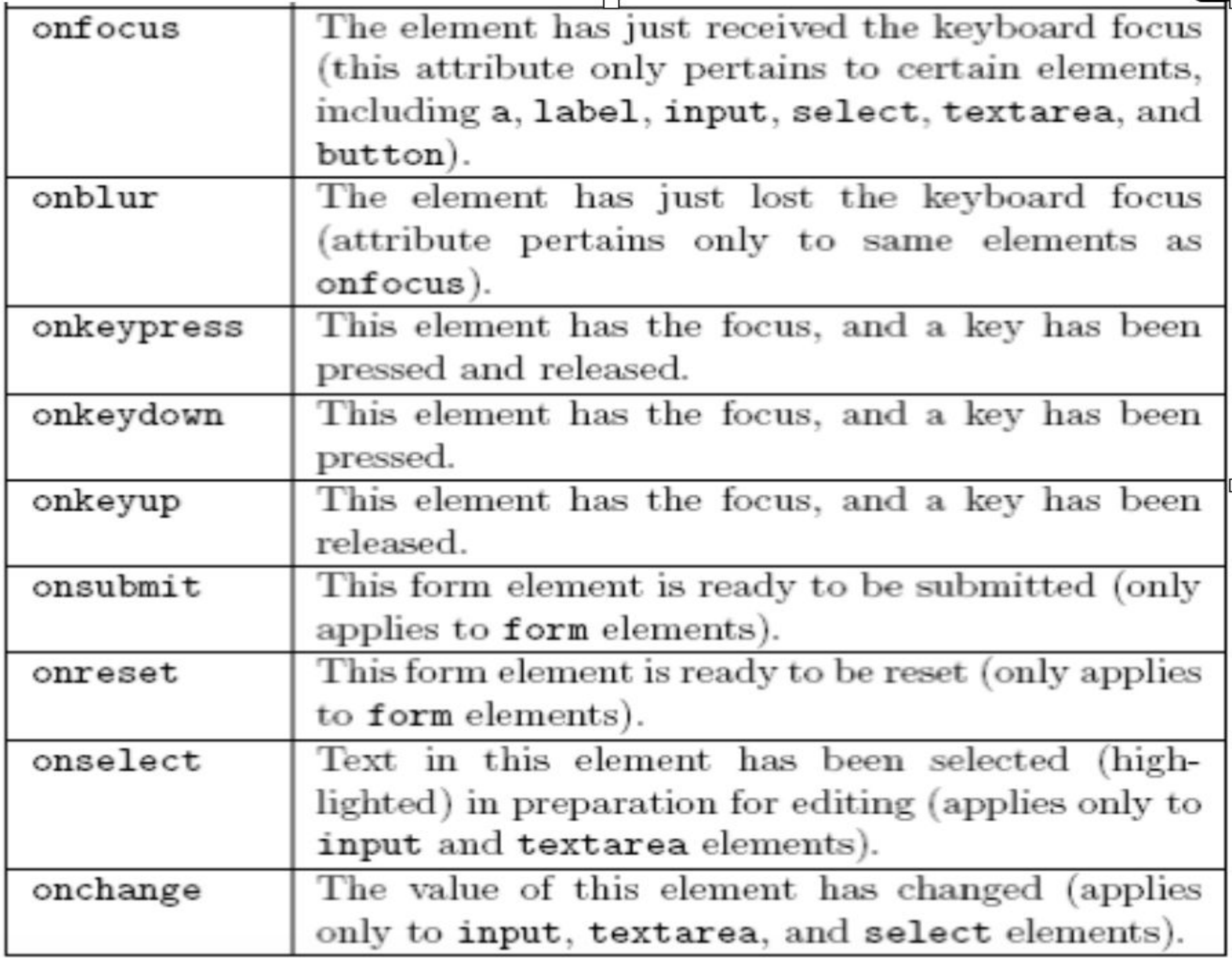
can be used to connect the events to handlers.

* Intrinsic Event Handling

TABLE 5.1: HTML intrinsic event attributes.



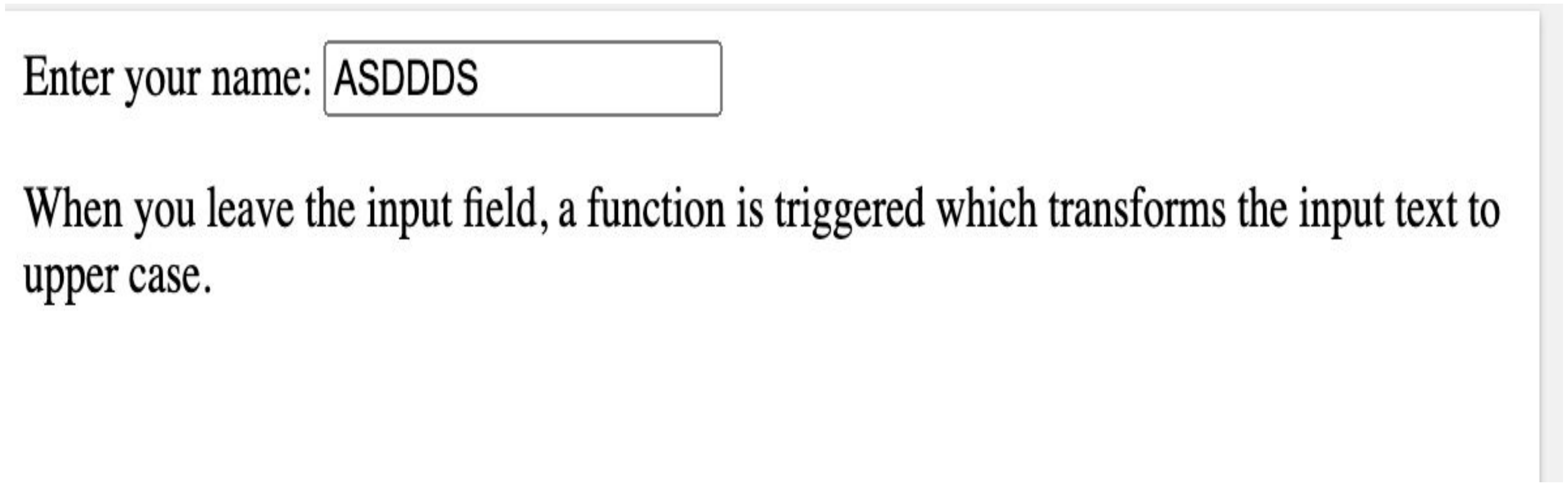
Attribute	When Called
onload	Immediately after the body of document has been fully read and parsed by the browser (this attribute only pertains to body and frameset).
onunload	The browser is ready to load a new document in place of the current document (this attribute only pertains to body and frameset).
onclick	A mouse button has been clicked and released over the element.
ondblclick	The mouse has been double-clicked over the element.
onmousedown	The mouse has been clicked over the element.
onmouseup	The mouse has been released over the element.
onmouseover	The mouse has just moved over the element.
onmousemove	The mouse has moved from one location to another over the element.
onmouseout	The mouse has just moved away from the element.



`onfocus`	The element has just received the keyboard focus (this attribute only pertains to certain elements, including `a`, `label`, `input`, `select`, `textarea`, and `button`).
`onblur`	The element has just lost the keyboard focus (attribute pertains only to same elements as `onfocus`).
`onkeypress`	This element has the focus, and a key has been pressed and released.
`onkeydown`	This element has the focus, and a key has been pressed.
`onkeyup`	This element has the focus, and a key has been released.
`onsubmit`	This form element is ready to be submitted (only applies to `form` elements).
`onreset`	This form element is ready to be reset (only applies to `form` elements).
`onselect`	Text in this element has been selected (highlighted) in preparation for editing (applies only to `input` and `textarea` elements).
`onchange`	The value of this element has changed (applies only to `input`, `textarea`, and `select` elements).

example of onblur

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

Enter your name: <input type="text" id="fname" onblur="myFunction()">

<p>When you leave the input field, a function is triggered which transforms the
input text to upper case.</p>

<script>
function myFunction() {
  var x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script>

</body>
</html>
```


Enter your name: ASDDDS

When you leave the input field, a function is triggered which transforms the input text to upper case.

REGISTERING AN EVENT HANDLER(three ways we will c)

- **There are different ways to register an event handler** in the DOM 0

First Way :

- event model. **One of these is by assigning the event handler script to an event tag attribute**, as in the following example:

```
            <input type = "button" id = "myButton“ onclick = "alert('You
clicked my button!');" />
```

- **Consider the following example of a button element:**

```
            <input type = "button" id = "myButton“  onclick
="myButtonHandler();" />
```

The handler consists of more than a single statement. In these cases, often a function is used and the literal string value of the attribute is the call to the function.

• Second way :of registering an Event Handler

- An event handler function could also be **registered by assigning its name to the associated event property on the button object,** as in the following example:
- **document.getElementById("myButton").onclick =myButtonHandler;**
- This statement must follow both the handler function and the form element so that JavaScript has seen both before assigning the property.
- The **name of the handler function is assigned to the property**—it is neither a string nor a call to the function.
- Third way is via addevent listener

Third way is ADDEVENTLISTENER

Explanation in the upcoming slides.

Example on the three way of registering an event is seen on the next slide.

Event handler example [WE CAN REGISTER AN EVENT BY USING ON[EVENT] -EG: ONCLICK OR ADDEVENTLISTENER

Dayananda Sagar University Bengaluru

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

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to attach a click event to a button.</p>

<!--button id="myBtn"  onclick=”displayDate();”     >Try it</button>-->

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

//document.getElementById("myBtn").onclick=displayDate;

document.getElementById("myBtn").addEventListener("click", displayDate);


function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
```

Following slides are examples on various events

Example :Handling Events from Text Box and Password Elements:

- **Text boxes and passwords** can create four different events: **blur, focus, change and select.**

For ex:

- Suppose JavaScript is used to compute the total cost of an order and display it to the customer before the order is submitted to the server for processing. An unscrupulous user may be tempted to change the total cost before submission, thinking that somehow an altered (and lower) price would not be noticed at the server end.
- Such a change to a text box can be prevented by an event handler **that blurs** the text box every time the user attempts to **put it in focus.**
- **Blur can be forced on an element with the blur method.**
- **The `HTMLElement.blur()` method removes keyboard focus from the current element. (Eg,this.blur())**

example on onfocus and onclick

```
<!DOCTYPE html>
<html lang = "en">
<head>
<title> nochange.html </title>
<meta charset = "utf-8" />
<script type = "text/javascript" src = "nochange.js" >
</script>
<style type = "text/css">
td, th, table {border: thin solid black}
</style>
</head>
<body>
<form action = "">
<h3> Coffee Order Form </h3>
```

```
<!-- A bordered table for item orders -->
<table>
<!-- First, the column headings -->
<tr>
<th> Product Name </th>
<th> Price </th>
<th> Quantity </th>
</tr>
<!-- Now, the table data entries -->
<tr>
<th> French Vanilla (1 lb.) </th>
<td> $3.49 </td>
```

```
<td> <input type = "text" id = "french"
size ="2" /> </td>
</tr>
<tr>
<th> Hazlenut Cream (1 lb.) </th>
<td> $3.95 </td>
<td> <input type = "text" id = "hazlenut"
size = "2" /> </td>
</tr>
<tr>
<th> Colombian (1 lb.) </th>
<td> $4.59 </td>
<td> <input type = "text" id = "colombian"
size = "2" /></td>
</tr>
</table>
<!-- Button for precomputation of the total cost -->
```

```
<p>
<input type = "button" value = "Total Cost"
onclick = "computeCost();" />
<input type = "text" size = "5" id = "cost"
onfocus = "this.blur();" />
</p>
<!-- The submit and reset buttons -->
<p>
<input type = "submit" value = "Submit Order" />
<input type = "reset" value = "Clear Order Form" />
</p>
</form>
</body>
```

nochange.js:

```
function computeCost() {
    var french = document.getElementById("french").value;
    var hazlenut = document.getElementById("hazlenut").value;
    var colombian = document.getElementById("colombian").value;
    document.getElementById("cost").value =
    totalCost = french * 3.49 + hazlenut * 3.95 +colombian *
    4.59;
}
```

```
<!DOCTYPE html>
<!-- pswd_chk.html
A document for pswd_chk.ps
Creates two text boxes for passwords
-->
<html lang = "en">
<head>
<title> Illustrate password checking> </title>
<meta charset = "utf-8" />
<script type = "text/javascript" src = "pswd_chk.js" >
</script>
</head>
<body>
<h3> Password Input </h3>
<form id = "myForm" action = "" >
<p>
```

example on password validation

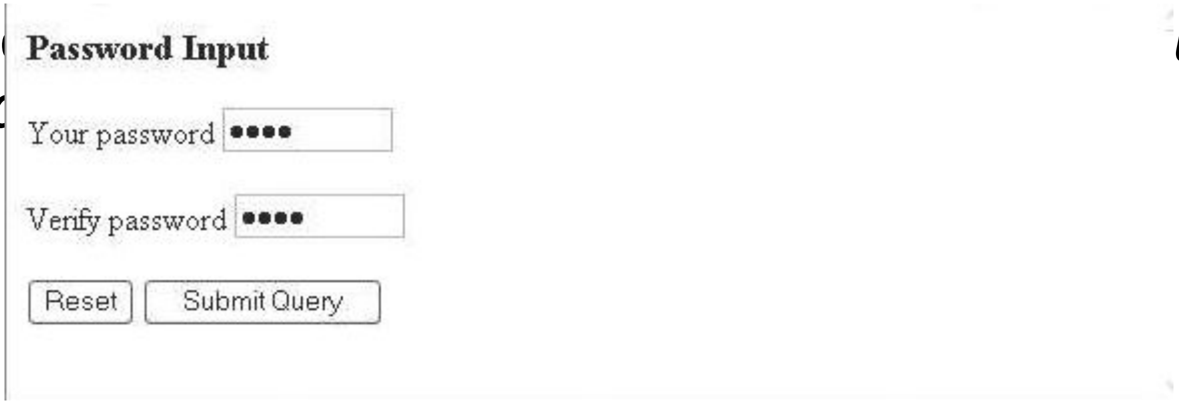
```
<label> Your password
<input type = "password" id = "initial"
size = "10" />
</label>
<br /><br />
<label> Verify password
<input type = "password" id = "second"
size = "10" />
</label>
<br /><br />
<input type = "reset" name = "reset" />
<input type = "submit" name = "submit" />
</p>
</form>
<!-- Script for registering the event handlers -->
<script type = "text/javascript" src = "pswd_chkr.js">
</script>
</body>
</html>
```

```
// pswd_chk.js
// An example of input password checking using the submit
// event
// The event handler function for password checking

function chkPasswords() {
var init = document.getElementById("initial");
var sec = document.getElementById("second");
if (init.value == "") {
alert("You did not enter a password \n" +
"Please enter one now");
return false;
}
if (init.value != sec.value) {
alert("The two passwords you entered are not the same \n" +
"Please re-enter both now");
return false;
} else
return true;
}
```

pswd_chkr.js

```
// Register the event handlers for pswd_chk.html

document.getElementById("second").onblur =
   chkPasswords;
document.getElementById("myForm").onsubmit =
   chkPasswords;
```

- Figure 1 shows a browser display of pswd_chk.html after the two password
  elem *uery has been clicked*



Password Input

Your password ••••

Verify password ••••

- Figure 2 shows a browser display that results from pressing the *Submit*
- *Query button on pswd_chk.html after different passwords have been entered*

onclick Event Type

- This is the most frequently used event type which occurs when a user clicks the left button of his mouse.

- One attribute can appear in several different tags:

  e.g. ***onClick*** can be in ***<a>*** and ***<input>***

- HTML element ***get focus:***

  1. When user puts mouse cursor over it and presses the left button
  2. When user tabs to the element
  3. By executing the ***focus*** method
  4. Element get blurred when another element gets focus

- **Event handlers can be specified two ways**
  1. Assigning the event handler script to an event tag attribute

     onClick = "alert('Mouse click!');"

     onClick = "myHandler();
  2. Assigning them to properties of JavaScript object associated with HTML elements.
- The ***load*** event: the completion of loading of a document by browser
- The ***onload*** attribute of ***<body>*** used to specify event handler:
- The ***unload*** event: used to clean up things before a document is unloaded.

Example:onclick

```
<!DOCTYPE html>
<html>
        <head>
                <script type="text/javascript">
                function sayHello(){}
                </script>
        </head>
        <body>
                <p>Click the following button and see result</p>
                <form>
                        <input type="button" onclick="sayHello()" value="Say Hello" />
                </form>
        </body>
</html>
```

onSubmit Event Type:

- onSubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.
- The following Next slide shows how to use onsubmit. Here we are calling a validate function before submitting a form data to the webserver. If validate function returns true, the form will be submitted, otherwise it will not submit the data.

onsubmit

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

function validate(){
   if ((document.example2.naming.value=="") || (document.example2.feed.value=="")){
      alert("You must fill in all of the required fields!")
      return false
   }
   else
      return true
}

</script>

<form name="example2" onsubmit="return validate()">

<input type="text" size="20" name="naming">
<strong>Feedback please: (*required)</strong>
<textarea name="feed" rows="3" cols="25"></textarea>
<strong>Your home address (*NOT required)</strong>
<input type="text" size="35" name="address">

<input type="submit" name="B1" value="Submit">
</form>
```

Some very important things here:

- `document.example2.naming.value==""`.What is the quotation in red? That is used to indicate an empty value- something that contains nothing. It is important that you distinguish between `""` and `" "` The later means "1 empty space", as opposed to "empty value". The later is a char- namely, a space.
- What is "`return true`", "`return false`"? This is what's used to actually allow, or stop the form from submitting, respectively. This is how JavaScript controls the submitting of a form. By default, a form will return true. (Submit the form). This is a important point- for example, by using the above knowledge, you can apply it to also stop a link from completing upon clicking. I'll show you an example:
  `<a href="http://www.cssdrive.com" onclick="return false">Click here, it won't work!</a>`
  Click here, it won't work!
  By `returning false`, we prohibit the action from completing!
- Now, a confusing point may be-`onsubmit="return validate()"`. Why `return validate()`? Wouldn't that be like a double return? No. Function validate only returns "true/false". You need "return true/fast" to actually manipulate whether a form submits or not. That's why we have to `return validate()`, as opposed to just `validate()`.

```
<!DOCTYPE html>
<html>
        <head>
                <script type="text/javascript">
                function validate()
                {
                 }
                </script>
        </head>
        <body>
                <form method="POST" action="target.html" onsubmit="return validate()">
                        .......
                        <input type="submit" value="Submit" />
                </form>
        </body>
</html>
```

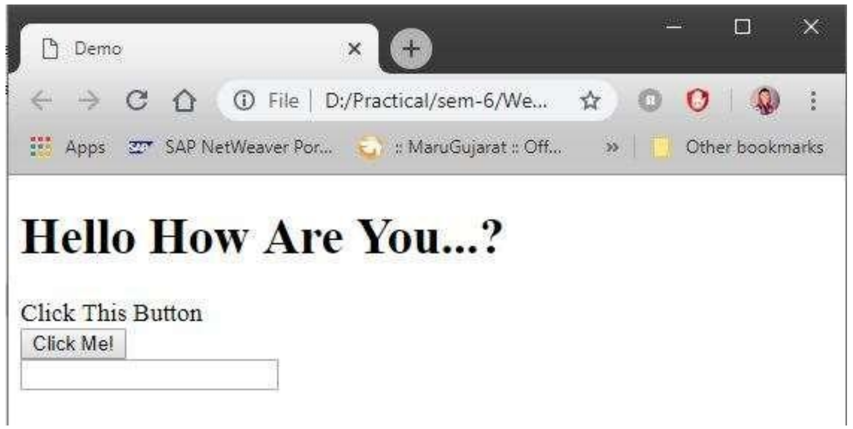
onmouseover and onmouseout :

- These two event types will help you create nice effects with images or even with text as well.
- The onmouseover event triggers when you bring your mouse over anyelement andthe onmouseout triggers when you move your mouse out from that element.
- Try the following example.

```
<html>

<body>
<p>Bring your mouse inside the division to see the result</p>
<div onmouseover="over()" onmouseout="out()">
<h2 id="l1"> This is inside the division </h2>
</div>

<script >
   function over()
   {
   document.getElementById("l1").innerHTML="Welcome ABC";
   }
  function out()
   {
   document.getElementById("l1").innerHTML="";
   }
 </script>
</body>
</html>
```

Focus & Blur Event Example:

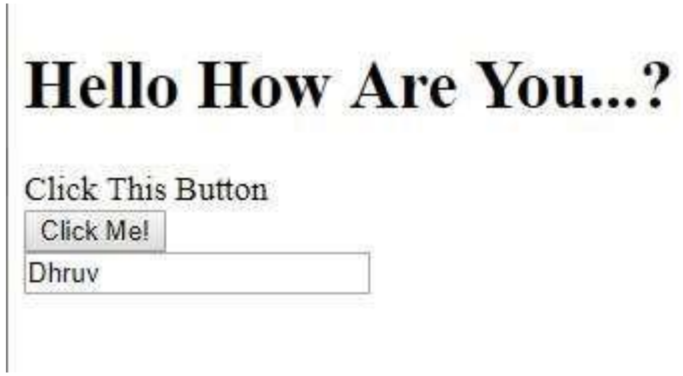
```
<!DOCTYPE html>
<html>
<head>
            <title>Demo</title>
</head>
<body>
            <h1>Hello How Are You...?</h1>
            <form>
                        Click This Button<br/>
                        <input type="button" value="Click Me!"
                        onclick="myFun()"/><br/>
                        <input type="text" id="username" onfocus="this.blur()"/><br/>
            </form>
            <script type="text/javascript">
                        function myFun()
                        {
           document.getElementById("username").value="Dhruv";
                        }
            </script>
</body>
</html>
```



Hello How Are You...?

Click This Button
Click Me!

[fig.1 Before Click On That Button]



Hello How Are You...?

Click This Button
Click Me!
Dhruv

[fig.2 After Click On That Button]

Third way of registering an event

3.addEventListener:

- The Event Target method **addEventListener()** sets up a function that

  will be called whenever the specified event is delivered to the target.

- Common targets are Element, Document, and Window, but the target may be any object that

  supports events (such as XML Http Request).
- addEventListener() works by adding a function or an object that implements

  Event Listener to the list of event listeners for the specified event type on the

  Event Target on which it's called.

The `addEventListener()` method attaches an event handler to the specified element.

The `addEventListener()` method attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the window object.

The `addEventListener()` method makes it easier to control how the event reacts to bubbling.

When using the `addEventListener()` method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

You can easily remove an event listener by using the `removeEventListener()` method.

Syntax

target.addEventListener(*type*, *listener[*, *options*]);

target.addEventListener(*type*, *listener[*, *useCapture*]);

target.addEventListener(*type*, *listener[*, *useCapture*, *wantsUntrusted*]);

Example:

document.getElementById("myBtn").addEventListener("click",

displayDate);

removeEventListener:

- The **EventTarget.removeEventListener()** method removes from the EventTarget an event listener previously registered with **EventTarget.addEventListener().**
- The event listener to be removed is identified using a combination of the event type, the event listener function itself, and various optional options that may affect the matching process; see Matching event listeners for removal.

Syntax

target.removeEventListener(*type*, *listener*[, *options*]);

target.removeEventListener(*type*, *listener*[,

useCapture]);

An Example of onclick

```
<!DOCTYPE html>
<html>
   <head><title>Display Page</title></head>
   <body>
      <hr color="orange" />
      <center><h1 id="htag">Welcome To ADIT</h1></center>
      <hr color="blue" />
      <center><button type="button" onclick="Change()">Change</button>
      <button type="button" onclick="Hide()">Hide</button>
      <button type="button" onclick="Display()">Display</button>
      <button type="button" onclick="ChangeColor()">Color Change</button></center>
      <hr color="green" />
      <script type="text/javascript">
      function Change()
         { document.getElementById("htag").innerHTML="Welcome ABC"; }


          function Display()
         { document.getElementById("htag").style.display="block"; }
         function Hide()
         { document.getElementById("htag").style.display="none"; }
         function ChangeColor()
         { document.getElementById("htag").style.color="blue"; }
      </script>
   </body>
</html>
```

Event handler example

revise :Event handler example [WE CAN REGISTER AN EVENT BY USING ON[EVENT] EG ONCLICK OR ADDEVENTLISTENER

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

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to attach a click event to a button.</p>

//<button id="myBtn"  onclick=”displayDate();”      >Try it</button>

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

//document.getElementById("myBtn").onclick=displayDate;

document.getElementById("myBtn").addEventListener("click", displayDate);


function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
</html>
```

using addEventListeneter calling two functions on the element

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

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to add two click events to the same button.</p>

<button id="myBtn">Try it</button>

<script>
var x = document.getElementById("myBtn");
x.addEventListener("click", myFunction);
x.addEventListener("click", someOtherFunction);

function myFunction() {
  alert ("Hello World!");
}

function someOtherFunction() {
  alert ("This function was also executed!");
}
</script>

</body>
</html>
```

addEventListener and removeEventListener

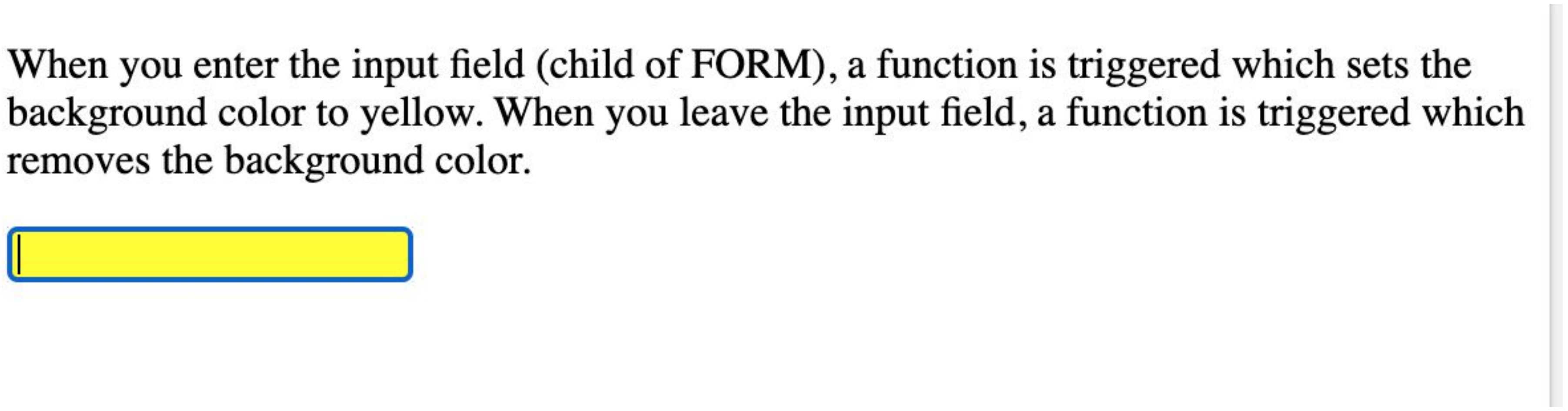
```
<!DOCTYPE html>
<html>
<head>
<style>
#myDIV {
  background-color: coral;
  border: 1px solid;
  padding: 50px;
  color: white;
  font-size: 20px;
}
</style>
</head>
<body>

<h2>JavaScript removeEventListener()</h2>

<div id="myDIV">
  <p>This div element has an onmousemove event handler that displays a random number every time you move your mouse inside this orange field.</p>
  <p>Click the button to remove the div's event handler.</p>
  <button onclick="removeHandler()" id="myBtn">Remove</button>
</div>

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

<script>
document.getElementById("myDIV").addEventListener("mousemove", myFunction);

function myFunction() {
  document.getElementById("demo").innerHTML = Math.random();
}

function removeHandler() {
  document.getElementById("myDIV").removeEventListener("mousemove", myFunction);
}
</script>

</body>
</html>
```

example :focus and blur on EventListener

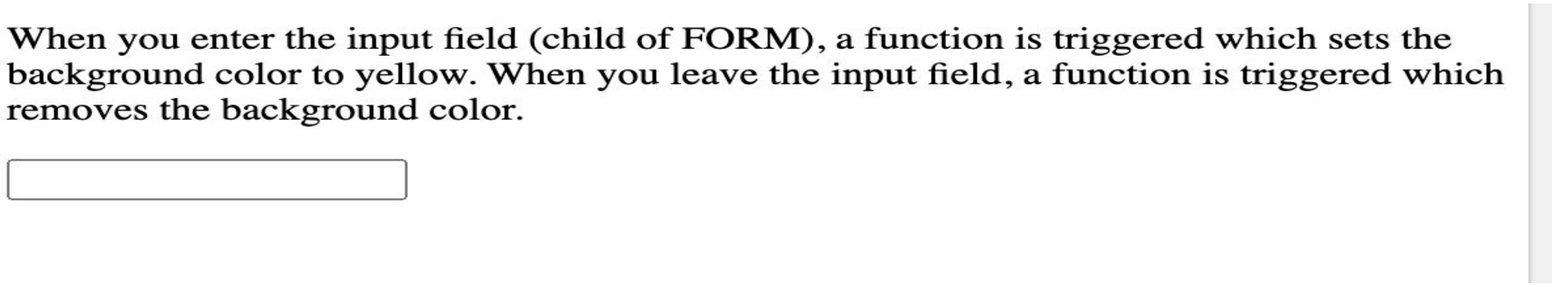
```
<!DOCTYPE html>
<html>
<body>

<p>When you enter the input field (child of FORM), a function is triggered which sets the background color to yellow. When you leave the
input field, a function is triggered which removes the background color.</p>

<form id="myForm">
  <input type="text" id="myInput">
</form>

<script>
var x = document.getElementById("myForm");
x.addEventListener("focus", myFocusFunction, true);
x.addEventListener("blur", myBlurFunction, true);

function myFocusFunction() {
  document.getElementById("myInput").style.backgroundColor = "yellow";
}

function myBlurFunction() {
  document.getElementById("myInput").style.backgroundColor = "";
}
</script>

</body>
</html>
```

Dayananda Sagar University Bengaluru

Output



When you enter the input field (child of FORM), a function is triggered which sets the background color to yellow. When you leave the input field, a function is triggered which removes the background color.



When you enter the input field (child of FORM), a function is triggered which sets the background color to yellow. When you leave the input field, a function is triggered which removes the background color.

addEventListener on window object

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

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method on the window object.</p>

<p>Try resizing this browser window to trigger the "resize" event handler.</p>

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

<script>
window.addEventListener("resize", function(){
  document.getElementById("demo").innerHTML = Math.random();
});
</script>

</body>
</html>
```

A brief Introduction on Regular Expressions

PATTERN MATCHING BY USING **REGULAR EXPRESSIONS :**

- JavaScript has powerful pattern-matching capabilities based on regular expressions.
- There are two approaches to pattern matching in JavaScript:

  i. RegExp object

  ii. Methods of the String object.
- The simplest pattern-matching method is **search**, which takes a pattern as a parameter.
- 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**.
- The position of the first character in the string is 0.

Reference

https://www.w3schools.com/js/js_regexp.asp

What is RegExp?

A regular expression is an object that describes a pattern of characters.When you search in a text, you can use a pattern to describe what you are searching for.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.Regular expressions are used to perform powerful pattern-matching and "search-and-replace" functions on text.

Syntax

var txt=new RegExp(pattern,modifiers);

or more simply:

var txt=/pattern/modifiers;

pattern specifies the pattern of an expression
modifiers specify if a search should be global, case-sensitive, etc.

RegExp Modifiers

Modifiers are used to perform case-insensitive and global searches.The i modifier is used to perform case-insensitive matching.The g modifier is used to perform a global match (find all matches rather than stopping after the first match).

```
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 />");
```

Output:

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

Dayananda Sagar University Bengaluru

……….end of module 3……….