FSD
Module-2

DEPT. OF COMPUTER SCIENCE & ENGINEERING,
DAYANANDA SAGAR UNIVERSITY,BENGALURU

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.

Object Orientation and JavaScript

- 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*

JavaScript Objects

- In JavaScript, objects are collections of properties, which correspond to the members of classes in Java and C++.
- Each property is either a data property 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

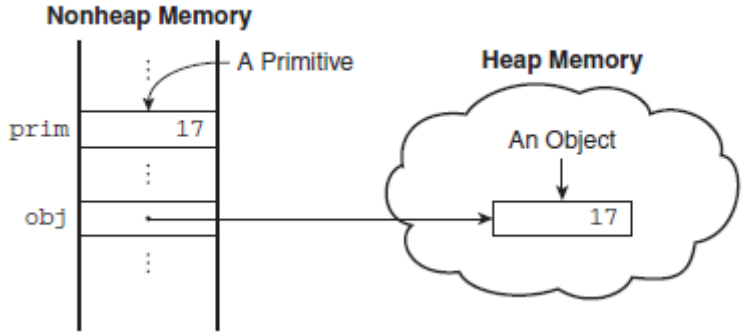
JavaScript has two forms of comments

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*



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,
index,
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,
d;
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

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

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the `new` keyword)
- Numbers can be objects (if defined with the `new` keyword)
- Strings can be objects (if defined with the `new` keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

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
	It determines whether the given value is a finite number.
	It determines whether the given value is an integer.
	It converts the given string into a floating point number.
	It converts the given string into an integer number.
	It returns the string that represents exponential notation of the given number.
	It returns the string that represents a number with exact digits after a decimal point.
	It returns the string representing a number of specified precision.
	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);

</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();
```

The String Catenation Operator

- 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 constructor.**

```
var number = Number(aString);
var number = aString - 0;
```

- The two, **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.

example of parseInt

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

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

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

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

<script>
function myFunction() {
  var a = parseInt("10") + "<br>";
  var b = parseInt("10.00") + "<br>";
  var c = parseInt("10.33") + "<br>";
  var d = parseInt("34 45 66") + "<br>";
  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;
  document.getElementById("demo").innerHTML = n;
}
</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.

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) = 'o'
- str.indexOf('r') = 3
- str.substring(2, 4) = 'org'
- str.toLowerCase() = 'george'

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;

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

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.

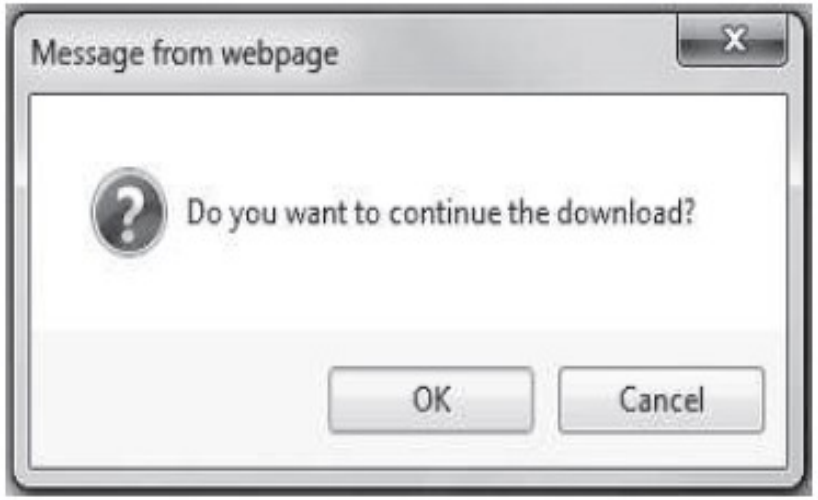
```
document.write("The result is: ", result, "<br />");
```

- The 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.

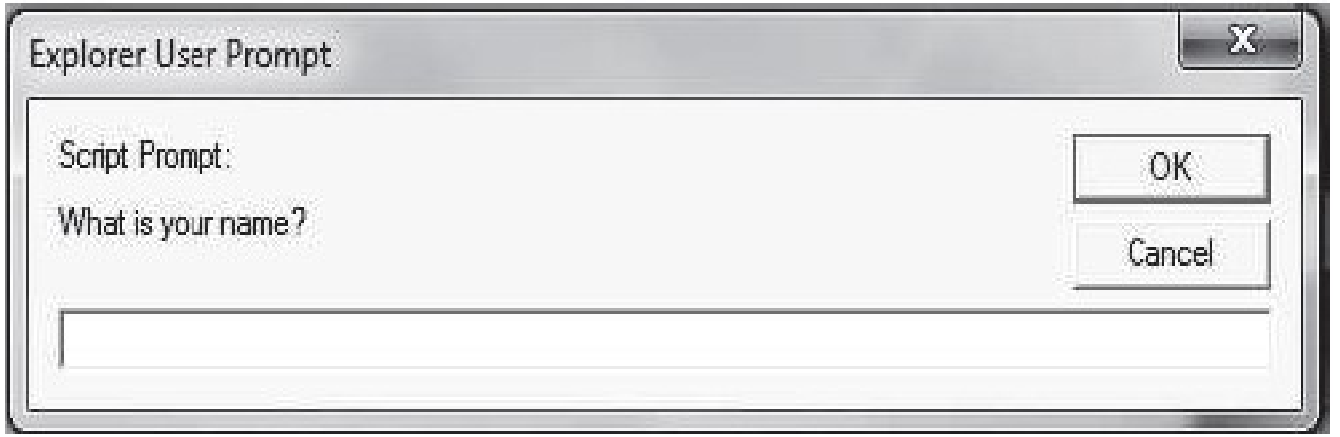
```
alert("The sum is:" + sum + "\n");
```

- **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?");



- **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>");
```

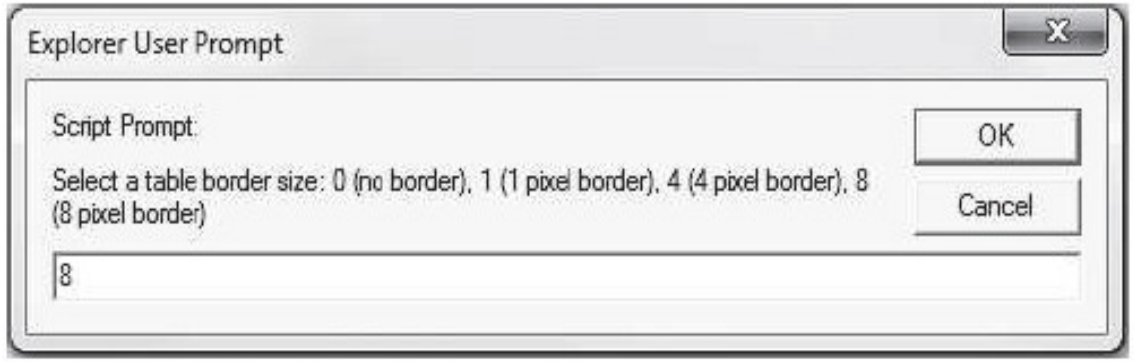


Figure 4.6 Dialog box from `borders2.js`

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;
```

```
// 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.

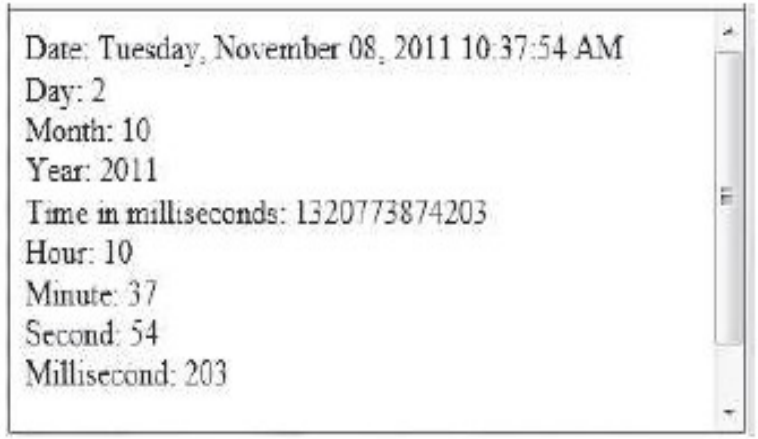


Figure 4.8 Display produced by `date.js`

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

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

```
<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)
{
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]

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.

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

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>
```

Array Methods

- **The join method converts all the elements of an array to strings and catenates them into a singl**e 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(" : ");
```

- **Reverse method:** It reverses the order of the elements of the Array object through which it is called.
- **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

```
<!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
```

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!
");}
- 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*.

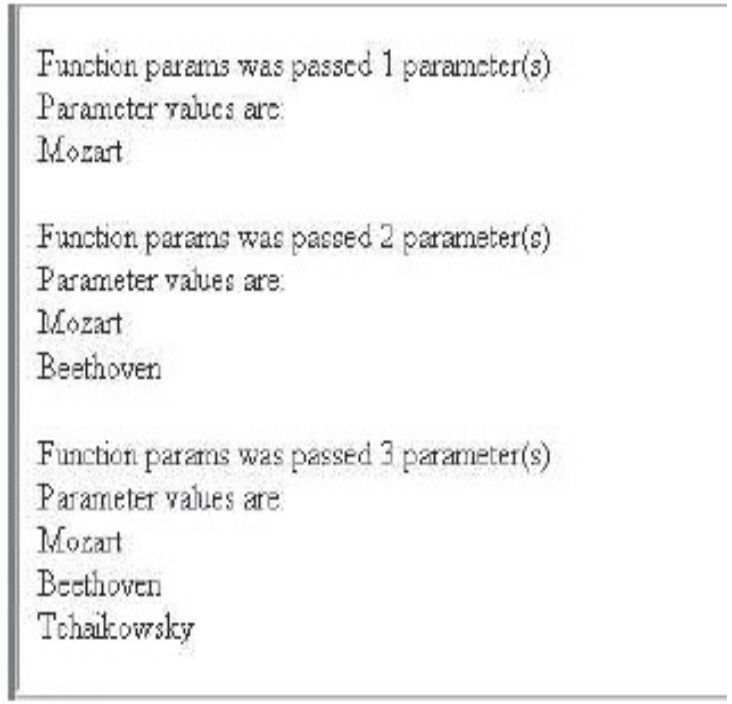
```
function fun1(my_list) {
var list2 = new Array(1, 3, 5);
my_list[3] = 14;
...
my_list = list2;
}
...
var list = new Array(2, 4, 6, 8)
fun1(list);
```

FUNCTION CAN BE STORED AS VARIABLE

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

<h2>JavaScript Functions</h2>
<p>A function can be stored in a variable:</p>
<p id="demo"></p>

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

</body>
</html>
```

```
<!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").innerHT
ML = x(4, 3);
</script>

</body>
</html>
```

```
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");
```



- 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];
```

SorT Method

- 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>
```

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").innerHTML =
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>
```

```
function median(list) {
list.sort(function (a, b) {return a - b;});
var list_len = list.length;
// Use the modulus operator to determine whether
// the array's length is odd or even
// Use Math.floor to truncate numbers
// Use Math.round to round numbers
if ((list_len % 2) == 1)
return list[Math.floor(list_len / 2)];
else
return Math.round((list[list_len / 2 - 1] +
list[list_len / 2]) / 2);
} // end of function median
// Test driver
var my_list_1 = [8, 3, 9, 1, 4, 7];
var my_list_2 = [10, -2, 0, 5, 3, 1, 7];
var med = median(my_list_1);
document.write("Median of [", my_list_1, "] is: ",
med, "<br />");
med = median(my_list_2);
document.write("Median of [", my_list_2, "] is: ",
med, "<br />");
```

```
Median of [1,3,4,7,8,9] is: 6
Median of [-2,0,1,3,5,7,10] is: 3
```

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.

```
function car(new_make, new_model, new_year) {
this.make = new_make;
this.model = new_model;
this.year = new_year;
}
```

could be used as in the following statement:

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

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

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

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

Table program using javascript

```
<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>
```