JavaScript in one page

Contents:
Review: Review, Allocation, Simple Examples;
JavaScript Language: Values, Data type conversion, Variables, Literals, Expressions, Operators, Statements, Functions, Built-in
Built-in Objects, Event;
Built-in JavaScript Objects: Root (Root Properties, Root Methods), Array (Array Description, Array Properties, Array Methods), B
Description), Data (Data Description, Data Methods), Function (Function Description), Image (Image Description, Image Properties), Ma
Math Properties, Math Methods), Number (Number Description, Number Properties), String (String Description, String Properties, String
Other: Other;
Similar Sites: MANUAL, Cheat sheets, HTML, CSS, XML, DTD, JavaScript, W3C DOM, SQL, SSI, Tell a friend, Free Icons, Itlibitum, C

Review

JavaScript is a compact, object-based scripting language for developing client and server Internet applications. Netscape Navigator interprets JavaScript statements embedded in an HTML page, and LiveWire enables you to create server-based applications similar to Common Gateway Interface (CGI) programs.

JavaScript is Netscape's cross-platform, object-based scripting language for client and server applications. There are two types of JavaScript:

- Navigator JavaScript, also called client-side JavaScript
- LiveWire JavaScript, also called server-side JavaScript

JavaScript is a language. Client and server JavaScript differ in numerous ways, but they have the following elements in common:

- Keywords, statement syntax, and grammar
- Rules for expressions, variables, and literals
- Underlying object model (although Navigator and LiveWire have different object frameworks)
- Built-in objects and functions

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitiona

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"

  <head>
    <title>... replace with your document's title ...</t
    <meta http-equiv="Content-Type" content="text/html;

    <script type="text/javascript" src="... insert link

    <script type="text/javascript">
      //<!— Begin to hide script contents from old brow
      ... or insert your JavaScript code here ...
      // End the hiding here. -->
    </script>

  </head>

  <body>
    ... replace with your document's content ...

    <script type="text/javascript">
      //<!— Begin to hide script contents from old brow
      ... or insert your JavaScript code here ...
      // End the hiding here. -->
    </script>

    ... replace with your document's content ...

    <input type="button" value="Click Me" onClick="... c

    <a href="... any link or sharp ..." onBlur="... or i

    ... replace with your document's content ...
  </body>

</html>
```

Hello World!

Code:

```
<html>
  <head></head>
  <body>
    <strong>Example:</strong>
    <script type="text/javascript">
      //<!—
      document.write("Hello
World!");
      //-->
    </script>
    <div>All done.</div>
  </body>
</html>
```

Example:

```
Hello World!
All done.
```

JavaScri

Defining and Dalling Func

Code:

```
<html>
  <head>
    <script type="text/javascript">
      //<!—
      function square(number) {
        return number * number;
      }
      //-->
    </script>
  </head>
  <body>
    <strong>Example:</strong>
    <script type="text/javascript">
      //<!—
      document.write("The function");
```

Example:

```
The function re
All done.
```

```
    document.write(" returned ");
    document.write(square(5), ".");
    //-->
  </script>
  <div>All done.</div>
 </body>
</html>
```

Values

JavaScript recognizes the following types of values:

- Numbers, such as 42 or 3.14159
- Logical (Boolean) values, either true or false
- Strings, such as "Howdy!"
- null, a special keyword denoting a null value

Data type conversion

JavaScript is a loosely typed language. That means you do not have to specify the data type of a variable when you declare it, and data types are converted automatically as needed during script execution. So, for example, you could define a variable as follows:

```
var answer = 42
```

And later, you could assign the same variable a string value, for example,

```
answer = "Thanks for all the fish..."
```

Because JavaScript is loosely typed, this assignment does not cause an error message.

In expressions involving numeric and string values, JavaScript converts the numeric values to strings. For example, consider the following statements:

```
x = "The answer is " + 42
y = 42 + " is the answer."
```

The first statement returns the string "The answer is 42." The second statement returns the string "42 is the answer."

You use var
rules.
A JavaScrip
sensitive, lett
Some exampl
You can dec

- By simply
- With the k

When you se
document. When
optional, but y
You can acc
variable called

You use literals to represent values in JavaScript. These are fixed values, not variables, that you literally provide in your sc

Integers

Integers can be expressed in decimal (base 10), hexadecimal (base 16), and octal (base 8). A decimal integer literal consists of a sequence of digits without a leading 0 (zero). A leading 0 (zero) on an integer literal indicates it is in octal; a leading 0x (or 0X) indicates hexadecimal. Hexadecimal integers can include digits (0-9) and the letters a-f and A-F. Octal integers can include only the digits 0-7. Some examples of integer literals are: 42, 0xFFF, and -345.

Floating-point literals

A floating-point literal can have the following parts: a decimal integer, a decimal point ("."), a fraction (another decimal number), an exponent, and a type suffix. The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-"). A floating-point literal must have at least one digit, plus either a decimal point or "e" (or "E"). Some examples of floating-point literals are 3.1415, -3.1E12, .1e12, and 2E-12

Boolean literals

The Boolean type has two literal values: true and false.

A string li
type; that is,
The followi
In addition

\b	
\f	
\n	
\r	
\t	
\\	

For charact
itself. You can

```
var quote = "
document.writ
```

The result
backslash chara

```
var home = "c
```

E

An expression is any valid set of literals, variables, operators, and expressions that evaluates to a single value; the value ca
Conceptually, there are two types of expressions: those that assign a value to a variable, and those that simply have a value.
For example, the expression x = 7 is an expression that assigns x the value seven.
This expression itself evaluates to seven. Such expressions use assignment operators. On the other hand, the expression 3 + 4 si
The operators used in such expressions are referred to simply as operators.
JavaScript has the following types of expressions:

- Arithmetic: evaluates to a number, for example 3.14159
- String: evaluates to a character string, for example, "Fred" or "234"
- Logical: evaluates to true or false

The special keyword null denotes a null value. In contrast, variables that have not been assigned a value are undefined and will
Array elements that have not been assigned a value, however, evaluate to false. For example, the following code executes the fur

```
myArray=new Array()
if (!myArray["notThere"])
myFunction()
```

A conditional expression can have one of two values based on a condition. The syntax is

```
(condition) ? val1 : val2
```

If condition is true, the expression has the value of val1. Otherwise it has the value of val2. You can use a conditional expres
For example,

```
status = (age >= 18) ? "adult" : "minor"
```

This statement assigns the value "adult" to the variable status if age is eighteen or greater. Otherwise, it assigns the value "

JavaScript has assignment, comparison, arithmetic, bitwise, logical, string, and special operators. This section describes the o
There are both binary and unary operators. A binary operator requires two operands, one before the operator and one after the op

```
operand1 operator operand2
```

For example, 3+4 or x*y.
A unary operator requires a single operand, either before or after the operator:

```
operator operand
```

or

```
operand operator
```

For example, x++ or ++x.

Assignment operators

An assignment operator assigns a value to its left operand based on the value of its right operand.
The basic assignment operator is equal (=), which assigns the value of its right operand to its left operand. That is, x = y assigns the value of y to x.
The other operators are shorthand for standard operations, as shown in the following table:

Shorthand operators

Shorthand operator	Meaning
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y
x <<= y	x = x << y
x >>= y	x = x >> y
x >>>= y	x = x >>> y
x &= y	x = x & y
x ^= y	x = x ^ y
x |= y	x = x | y

Comparison operators

A comparison operator compares its operands and r
based on whether the comparison is true or not.
The operands can be numerical or string values. Wher
the comparisons are based on the standard lexicographica
They are described in the following table.

Comparison operators

Operator	Name	Description	Exa
==	Equal	Returns true if the operands are equal	x
!=	Not equal	Returns true if the operands are not equal.	x
>	Greater than	Returns true if left operand is greater than right operand.	x
>=	Greater than or equal	Returns true if left operand is greater than or equal to right operand.	x
<	Less than	Returns true if left operand is less than right operand.	x
<=	Less than or equal	Returns true if left operand is less than or equal to right operand.	x

Arithmetic operators

Arithmetic operators take numerical values (either literals or variables) as their operands and return a single numerical value.
The standard arithmetic operators are addition (+), subtraction (-), multiplication (*), and division (/).
These operators work as they do in other programming languages.

Arithmetic operators

Synopsis	Name	Description	Example	Example Description
var1 % var2	Modulus	Returns the first operand modulo the second operand, that is, var1 modulo var2, in the preceding statement, where var1 and var2 are variables. The modulo function is the floating-point remainder of dividing var1 by var2	13 % 5	Returns 3
var++	Increment	Increments (adds one to) its operand and returns a value. If used postfix, with operator after operand (for example, x++), then it returns the value before incrementing. If used prefix with operator before operand (for example, ++x), then it returns the value after incrementing.	y = x++	If x is three, then the statement y = x++ sets y to three and increments x to four.
++var			y=++x	If x is three, then the statement y = ++x increments x to four and sets y to four.
var--	Decrement	Decrements (subtracts one from) its operand and returns a value. If used postfix (for example, x--), then it returns the value before decrementing. If used prefix (for example, --x), then it returns the value after decrementing.	y = x--	If x is three, then the statement y = x-- sets y to three and decrements x

Bitwise ope
For example
they return sta

Operator	
Bitwise AND	a
Bitwise OR	a
Bitwise XOR	a
Bitwise NOT	~
Left shift	a
Sign-propagating right shift	a
Zero-fill right shift	a

				to two.
--var			y=—x	If x is three, then the statement y = --x decrements x to two and sets y to two.
-var	Unary negation	The unary negation precedes its operand and negates it.	x = -x	Negates the value of x; that is, if x were three, it would become -3.

String operators

In addition to the comparison operators, which can be used on string values, the concatenation operator (+) concatenates two string values together, returning another string that is the union of the two operand strings. For example, "my " + "string" returns the string "my string".

The shorthand assignment operator += can also be used to concatenate strings. For example, if the variable mystring has the value "alpha," then the expression mystring += "bet" evaluates to "alphabet" and assigns this value to mystring.

Special operators

Name	Synopsis	Description	Exam
new	objectName = new objectType (param1 [,param2] ...[,paramN])	You can use the new operator to create an instance of a user-defined object type or of one of the built-in object types Array, Boolean, Date, Function, Math, Number, or String.	
typeof	typeof operand	The typeof operator returns a string indicating the type of the unevaluated operand. operand is the string, variable, keyword, or object for which the type is to be returned. The parentheses are optional.	Suppose you define the var myFun = new Functior var shape="round" var size=1 var today=new Date() The typeof operator retu results for these variab typeof myFun is object typeof shape is string typeof size is number typeof today is object typeof dontExist is unde
	typeof (operand)		
void	javascript:void (expression)	The void operator specifies an expression to be evaluated without returning a value. expression is a JavaScript expression to evaluate. The parentheses surrounding the expression are optional, but it is good style to use them. You can use the void operator to specify an expression as a hypertext link. The expression is evaluated but is not loaded in place of the current document.	<a href="javascript:voi Click here to do nothing
	javascript:void expression		<a href="javascript:void(do Click here to submit

Statements (

break

break

A statement that terminates the current while or program control to the statement following the terminate

```
1. // comment text
2. /* multiple line comment text */
```

comment

Notations by the author to explain what a script do by the interpreter. JavaScript supports Java-style comme

- Comments on a single line are preceded by a dou
- Comments that span multiple lines are preceded a */.

continue

continue

A statement that terminates execution of the block or for loop, and continues execution of the loop with contrast to the break statement, continue does not ter the loop entirely: instead,

- In a while loop, it jumps back to the condition.
- In a for loop, it jumps to the update expression.

```
for ([initial-expression;] [condition;] [increment-expression]) {
  statements
}
```

for

A statement that creates a loop that consis expressions, enclosed in parentheses and separated by se

block of statements executed in the loop.

Arguments

- initial-expression is a statement or variable decl
  used to initialize a counter variable. This exp
  declare new variables with the var keyword.
- condition is evaluated on each pass through the
  evaluates to true, the statements in statement
  conditional test is optional. If omitted, the con
  to true.
- increment-expression is generally used to update c
  variable.
- statements is a block of statements that are execut
  evaluates to true. This can be a single statement
  Although not required, it is good practice to inde
  the beginning of the for statement.

```
for (variable in object) {
statements
}
```

for...in
A statement that iterates a specified variable ove
an object. For each distinct property, JavaScript
statements.

Arguments

- variable is the variable to iterate over every prop
- object is the object for which the properties are i
- statements specifies the statements to execute for

```
function name([param] [, param] [..., param]) {
statements
}
```

function
A statement that declares a JavaScript function n
parameters param. Acceptable parameters include strings,
To return a value, the function must have a return s
the value to return. You cannot nest a function stateme
or in itself.
All parameters are passed to functions, by value. If
is passed to the function, but if the function cha
parameter, this change is not reflected globally or in t

Arguments

- name is the function name.
- param is the name of an argument to be passed to t
  can have up to 255 arguments.

```
if (condition) {
statements1
} [else {
statements2
}]
```

if...else
A statement that executes a set of statements if a
true. If the condition is false, another set of statemen

Arguments

- condition can be any JavaScript expression that
  false. Parentheses are required around the co
  evaluates to true, the statements in statements1 ar
- statements1 and statements2 can be any JavaScrip
  further nested if statements. Multiple statement
  braces.

```
objectName = new objectType ( param1 [,param2] ...[,paramN] )
```

new
An operator that lets you create an instance of a u
or of one of the built-in object types Array, Boolean,
Number, or String.
Creating a user-defined object type requires two ste

1. Define the object type by writing a function.
2. Create an instance of the object with new.

To define an object type, create a function for
specifies its name, properties, and methods. An object c
is itself another object. See the examples below.
You can always add a property to a previously defin
the statement car1.color = "black" adds a property col
it a value of "black". However, this does not affect an
the new property to all objects of the same type, you m
the definition of the car object type.

Arguments

- objectName is the name of the new object instance.
- objectType is the object type. It must be a fu
  object type.
- param1...paramN are the property values for the o
  are parameters defined for the objectType function.

```
 return expression
```

return
    A statement that specifies the value to be returned

```
switch(variable) {
  case value_1:
    statements_1;
    break;
  case value_1:
    statements_2;
    break;
...
  default:
    statements_default;
}
```

switch
    The switch statement can be used for multiple branch
string.

Arguments:

- variable is any variable.
- value_... is any valid value.
- statements_... is any block of statements.

```
 this[.propertyName]
```

this
    A keyword that you can use to refer to the current
method this refers to the calling object.

```
 var varname [= value] [..., varname [= value] ]
```

var
    A statement that declares a variable, optionally
value. The scope of a variable is the current func
declared outside a function, the current application.
    Using var outside a function is optional; you can
simply assigning it a value. However, it is good style
necessary in functions if a global variable of the same

Arguments

- varname is the variable name. It can be any legal i
- value is the initial value of the variable
  expression.

```
while (condition) {
  statements
}
```

while
    A statement that creates a loop that evaluates an e
true, executes a block of statements. The loop then r
specified condition is true.

Arguments

- condition is evaluated before each pass throu
  condition evaluates to true, the statements in th
  performed. When condition evaluates to false, execu
  statement following statements.
- statements is a block of statements that are ex
  condition evaluates to true. Although not required,
  indent these statements from the beginning of the w

```
do {
  statements
} while (condition)
```

```
with (object) {
  statements
}
```

with
    A statement that establishes the default object fo
Within the set of statements, any property references
object are assumed to be for the default object.

Arguments

- object specifies the default object to use fo
  parentheses around object are required.
- statements is any block of statements.

Functions are one of the fundamental building blocks in JavaScript. A function is a JavaScript procedure - a set of statements t

Defining functions

A function definition consists of the function keyword, followed by

- The name of the function.
- A list of arguments to the function, enclosed in parentheses and separated by commas.
- The JavaScript statements that define the function, enclosed in curly braces, { }. The statements in a function can include calls to other functions defined in the current application.

In Navigator JavaScript, it is good practice to define all your functions in the HEAD of a page so that when a user loads the page, the functions are loaded first.

For example, here is the definition of a simple function named pretty_print:

```
function pretty_print(str) {
  document.write("<hr><p>" + str)
}
```

, or semantically equivalent:

```
var pretty_print = function(str) {
  document.write("<hr><p>" + str)
}
```

This function takes a string, str, as its argument, adds some HTML tags to it using the concatenation operator (+), and then displays the result to the current document using the write method.

Using functions

In a Navigator application, you can use (or call) the current page. You can also use functions defined by frames. In a LiveWire application, you can use any fun application.

Defining a function does not execute it. You have t it to do its work. For example, if you defined pretty_print in the HEAD of the document, you could call

```
<script type="text/javascript">
  pretty_print("This is some text to display")
</script>
```

The arguments of a function are not limited to strir pass whole objects to a function, too.

A function can even be recursive, that is, it can ca here is a function that computes factorials:

```
function factorial(n) {
  if ((n == 0) || (n == 1)) return 1
  else {
    result = (n * factorial(n-1) )
    return result
  }
}
```

You could then display the factorials of one through

```
for (x = 0; x < 5; x++) {
  document.write("<br />", x, " factorial is ", factor
}
```

The results are:

```
0 factorial is 1
1 factorial is 1
2 factorial is 2
3 factorial is 6
4 factorial is 24
5 factorial is 120
```

Buil

`isNaN(testValue)`

isNaN function

The isNaN function evaluates an argument to determir number).

Arguments

- testValue is the value you want to evaluate.

On platforms that support NaN, the parseFloat and pa "NaN" when they evaluate a value that is not a number. passed "NaN," and false otherwise.

`parseFloat(str)`

parseFloat

parseFloat parses its argument, the string str, ar floating-point number. If it encounters a character othe a numeral (0-9), a decimal point, or an exponent, then to that point and ignores that character and all succee first character cannot be converted to a number, it number).

`parseInt(str [, radix])`

parseInt

parseInt parses its first argument, the string str, an integer of the specified radix (base), indicated b argument, radix. For example, a radix of ten indicates number, eight octal, sixteen hexadecimal, and so on. For letters of the alphabet indicate numerals greater than hexadecimal numbers (base 16), A through F are used.

If parseInt encounters a character that is not a n radix, it ignores it and all succeeding characters a value parsed up to that point. If the first character c number in the specified radix, it returns "NaN." truncates numbers to integer values.

JavaScript is based on a simple object-oriented paradigm.
An object is a construct with properties that are JavaScript variables or other objects.
An object also has functions associated with it that are known as the object's methods.
In addition to objects that are built into the Navigator client and the LiveWire server, you can define your own objects.

Creating new objects

Both client and server JavaScript have a number of predefined objects. In addition, you can create your own objects. Creating your own object requires two steps:

1. Define the object type by writing a constructor function.
2. Create an instance of the object with new.

To define an object type, create a function for the object type that specifies its name, properties, and methods. For example, suppose you want to create an object type for cars. You want this type of object to be called car, and you want it to have properties for make, model, year, and color. To do this, you would write the following function:

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

Notice the use of this to assign values to the object's properties based on the values passed to the function.
Now you can create an object called mycar as follows:

```
mycar = new car("Eagle", "Talon TSi", 1993)
```

This statement creates mycar and assigns it the specified values for its properties. Then the value of mycar.make is the string "Eagle," mycar.year is the integer 1993, and so on.
You can create any number of car objects by calls to new. For example,

```
kenscar = new car("Nissan", "300ZX", 1992)
```

An object can have a property that is itself another object. For example, suppose you define an object called person as follows:

```
function person(name, age, sex) {
  this.name = name;
  this.age = age;
  this.sex = sex;
}
```

and then instantiate two new person objects as follows:

```
rand = new person("Rand McKinnon", 33, "M")
ken = new person("Ken Jones", 39, "M")
```

Then you can rewrite the definition of car to include an owner property that takes a person object, as follows:

```
function car(make, model, year, owner) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
}
```

To instantiate the new objects, you then use the following:

```
car1 = new car("Eagle", "Talon TSi", 1993, rand)
car2 = new car("Nissan", "300ZX", 1992, ken)
```

Notice that instead of passing a literal string or integer value when creating the new objects, the above statements pass the objects rand and ken as the arguments for the owners. Then if you want to find out the name of the owner of car2, you can access the following property:

```
car2.owner.name
```

Note that you can always add a property to a previously defined object. For example, the statement

```
car1.color = "black"
```

adds a property color to car1, and assigns it a value of "black." However, this does not affect any other objects. To add the new property to all objects of the same type, you have to add the property to the definition of the car object type.

A method is
associate the f

```
object.methodn
```

where objec
You can the

```
object.methodn
```

You can def
that would form

```
function disp
  var result =
  pretty_prin
}
```

where prett
which the metho
You can mak

```
this.displayCa
```

to the obje

```
function car(m
  this.model =
  this.year =
  this.owner =
  this.displa
}
```

Then you ca

```
car1.displayCa
car2.displayCa
```

This will p

```
A Beautiful 19
A Beautiful 19
```

Defining object with "return"

Let's consider a person object with first and last name fields. There are two ways in which their name might be displayed: as "first

```
function Person(first, last) {
  return {
    first: first,
    last: last,
    fullName: function() {
      return this.first + ' ' + this.last;
    },
    fullNameReversed: function() {
      return this.last + ', ' + this.first;
    }
```

Defining object with "th

```
function Person(first, last) {
  this.first = first;
  this.last = last;
  this.fullName = function() {
    return this.first + ' ' + this.last;
  }
  this.fullNameReversed = function() {
    return this.last + ', ' + this.first;
  }
}
```

```
    }
  }
```

use (with trace):

```
> s = Person("Simon", "Willison")
> s.fullName()
Simon Willison
> s.fullNameReversed()
Willison, Simon
```

, or:

```
function personFullName() {
    return this.first + ' ' + this.last;
  }
function personFullNameReversed() {
    return this.last + ', ' + this.first;
  }

function Person(first, last) {
  this.first = first;
  this.last = last;
  this.fullName = personFullName
  this.fullNameReversed = personFullNameReversed
}
```

use (with trace):

```
> s = new Person("Simon", "Willison")
> s.fullName()
Simon Willison
> s.fullNameReversed()
Willison, Simon
```

Bui

JavaScript Root Object Properties (for all built-in objects)

constructor	A reference to the function that created the object	eval(string)

Example №1
In this example we will show how to use the constructor property:

```
var test=new Array();
if (test.constructor==Array) {document.write("This is an Array")};
if (test.constructor==Boolean) {document.write("This is a Boolean")};
if (test.constructor==Date) {document.write("This is a Date")};
if (test.constructor==String) {document.write("This is a String")}
```

The output of the code above will be:

```
This is an Array
```

Example 2 toSource()
In this example we will show how to use the constructor property:

```
function employee(name,jobtitle,born) {this.name=name; this.jobtitle=jobtitle; this.born=born;}
var fred=new employee("Fred Flintstone","Caveman",1970);
document.write(fred.constructor);
```

The output of the code above will be:

```
function employee(name,jobtitle,born) { this.name=name; this.jobtitle=jobtitle ; this.born=born; }
```

prototype	Lets you add a properties to an object.	toString()

Example
In this example we will show how to use the prototype property to add a property to an object:

```
function employee(name,jobtitle,born) {this.name=name; this.jobtitle=jobtitle; this.born=born;}
var fred=new employee("Fred Flintstone","Caveman",1970);
employee.prototype.salary=null;
fred.salary=20000;
document.write(fred.salary);
```

valueOf()

The output of the code above will be:

```
20000
```

JavaScript Array Object Description

Review
JavaScript does not have an explicit array data type. However, you can use the built-in Array object and its methods to work with arrays in your applications. The Array object has methods for joining, reversing,

length	Reflects the nu

and sorting arrays. It has a property for determining the array length.

An array is an ordered set of values that you reference through a name and an index. For example, you could have an array called emp that contains employees' names indexed by their employee number. So emp[1] would be employee number one, emp[2] employee number two, and so on.

To create an Array object:

1. arrayObjectName = new Array([arrayLength])
2. arrayObjectName = new Array([element0, element1, ..., elementn])

Arguments

- arrayObjectNameis either the name of a new object or a property of an existing object. When using Array properties and methods, arrayObjectName is either the name of an existing Array object or a property of an existing object.
- arrayLengthis the initial length of the array. You can access this value using the length property.
- elementnis a list of values for the array's elements. When this form is specified, the array is initalized with the specified values as its elements, and the array's length property is set to the number of arguments.

The Array object has the following main methods:

- join - joins all elements of an array into a string
- reverse - transposes the elements of an array: the first array element becomes the last and the last becomes the first
- sort - sorts the elements of an array

For example, suppose you define the following array:

```
myArray = new Array("Wind","Rain","Fire")
```

myArray.join() returns "Wind,Rain,Fire"; myArray.reverse transposes the array so that myArray[0] is "Fire", myArray[1] is "Rain", and myArray[2] is "Wind". myArray.sort sorts the array so that myArray[0] is "Fire", myArray[1] is "Rain", and myArray[2] is "Wind". myArray.

Defining Arrays

The Array object is used to store a set of values in a single variable name.

We define an Array object with the new keyword. The following code line defines an Array object called myArray:

```
var myArray=new Array()
```

There are two ways of adding values to an array (you can add as many values as you need to define as many variables you require).

1:

```
var mycars=new Array();
mycars[0]="Saab";
mycars[1]="Volvo";
mycars[2]="BMW"
```

You could also pass an integer argument to control the array's size:

```
var mycars=new Array(3);
mycars[0]="Saab";
mycars[1]="Volvo";
mycars[2]="BMW"
```

2:

```
var mycars=new Array("Saab","Volvo","BMW")
```

Note:

If you specify numbers or true/false values inside the array then the type of variables will be numeric or Boolean instead of string.

Accessing Arrays

You can refer to a particular element in an array by referring to the name of the array and the index number. The index number starts at 0.

The following code line:

```
document.write(mycars[0])
```

will result in the following output:

```
Saab
```

Modify Values in Existing Arrays

To modify a value in an existing array, just add a new value to the array with a specified index number:

```
mycars[0]="Opel"
```

concat (arrayX,arrayX,.....,arrayX)

Joins two returns the resu

This metho existing arrays, of the joined a

Arguments

- arrayX - o to be join

eval(string)

Evaluates a

- string is existing ol

join(separator)

Joins all e a string.

The string elements are jo

- separator separate array. The to a str omitted, t separated

pop()

Removes ar element of an a

The pop() m and return th array.

Note:

This method the array.

Tip:

To remove element of an method.

push()

Adds one or

The push() length.

Arguments

- newelement
- newelement
- newelement

Note:

This method

Tip:

To add one

reverse()

Transposes array: the firs the last and first.

The reverse elements of the

shift()

Removes an

Now, the following code line:

```
document.write(mycars[0])
```

will result in the following output:

```
Opel
```

Two-dimensional array

The following code creates a two-dimensional array and displays the results.

```
a = new Array(4);
for (i=0; i < 4; i++) {
  a[i] = new Array(4);
  for (j=0; j < 4; j++) {
    a[i][j] = "["+i+","+j+"]";
  }
}
for (i=0; i < 4; i++) {
  str = "Row "+i+":";
  for (j=0; j < 4; j++) {
    str += a[i][j];
  }
document.write(str,"<p>")
}
```

This example displays the following results:

```
Multidimensional array test
Row 0:[0,0][0,1][0,2][0,3]
Row 1:[1,0][1,1][1,2][1,3]
Row 2:[2,0][2,1][2,2][2,3]
Row 3:[3,0][3,1][3,2][3,3]
```

element of an a
The shift(
remove and retu
an array.
Note:
This method
the array.
Tip:
To remove
element of an
method.

slice(start,end)

Returns sel
existing array
The slic
selected eleme
array.

Arguments:

1. start (Req
start the
number
2. end (Optic
end the
number

Note:
If end is
selects all ele
start position
array.
Tip:
You can us
select from the

sort(sortby)

Sorts the e
The sort()
the elements of

Argument

- sortby (O
sort order.

Note:

- The sort()
elements
default. H
numbers
correctly
sort numbe
function t
- After usin
array is c

splice (index,howmany,element1,.....,elementX)

Removes and
array
The splice
remove and ad
array.

Arguments:

- index (Req
add/remove
number
- howmany (
many eleme
Must be a

- element1 (
  element to
- elementX
  elements c

unshift (newelement1,newelement2, ...,newelementX)	Adds one or The unshift new length. Arguments: • newelement • newelement • newelement Note: • This metho • The unshif Tip: • To add one

JavaScript Boo

Review

Use the built-in Boolean object when you need to convert a non-boolean value to a boolean value. You can use the Boolean object the valueOf method.

To create a Boolean object:

```
var booleanObjectName = new Boolean(value)
```

- booleanObjectName is either the name of a new object or a property of an existing object. When using Boolean properties, boolea
- value is the initial value of the Boolean object. The value is converted to a boolean value, if necessary. If value is omitt create an object with an initial value of true.

The following examples create Boolean objects:

```
bfalse = new Boolean(false)
btrue = new Boolean(true)
```

All the following lines of code create Boolean objects with an initial value of false:

```
var myBoolean=new Boolean(); var myBoolean=new Boolean(0); var myBoolean=new Boolean(null)
var myBoolean=new Boolean(""); var myBoolean=new Boolean(false); var myBoolean=new Boolean(NaN)
```

And all the following lines of code create Boolean objects with an initial value of true:

```
var myBoolean=new Boolean(true); var myBoolean=new Boolean("true")
var myBoolean=new Boolean("false"); var myBoolean=new Boolean("Richard")
```

JavaScript Data Object Description

Review

JavaScript does not have a date data type. However, you can use the Date object and its methods to work with dates and times in your applications. The Date object has a large number of methods for setting, getting, and manipulating dates. It does not have any properties.

JavaScript handles dates similarly to Java. The two languages have many of the same date methods, and both languages store dates as the number of milliseconds since January 1, 1970, 00:00:00.

Note

Currently, you cannot work with dates prior to January 1, 1970.

To create a Date object:

```
dateObjectName = new Date([parameters])
```

Date()	Returns tod
getDate()	Returns the Date object (fr

where dateObjectName is the name of the Date object being created; it can be a new object or a property of an existing object.

The parameters in the preceding syntax can be any of the following:

- Nothing: creates today's date and time. For example, today = new Date().
- A string representing a date in the following form: "Month day, year hours:minutes:seconds." For example, Xmas95 = new Date("December 25, 1995 13:30:00"). If you omit hours, minutes, or seconds, the value will be set to zero.
- A set of integer values for year, month, and day. For example, Xmas95 = new Date(95,11,25). A set of values for year, month, day, hour, minute, and seconds. For example, Xmas95 = new Date(95,11,25,9,30,0)

Methods of the Date object

The Date object methods for handling dates and times fall into these broad categories:

- "set" methods, for setting date and time values in Date objects.
- "get" methods, for getting date and time values from Date objects.
- "to" methods, for returning string values from Date objects.
- parse and UTC methods, for parsing Date strings.

With the "get" and "set" methods you can get and set seconds, minutes, hours, day of the month, day of the week, months, and years separately. There is a getDay method that returns the day of the week, but no corresponding setDay method, because the day of the week is set automatically.

These methods use integers to represent these values as follows:

- Seconds and minutes: 0 to 59
- Hours: 0 to 23
- Day: 0 to 6 (day of the week)
- Date: 1 to 31 (day of the month)
- Months: 0 (January) to 11 (December)
- Year: years since 1900

For example, suppose you define the following date:

```
Xmas95 = new Date("December 25, 1995")
```

Then Xmas95.getMonth() returns 11, and Xmas95.getYear() returns 95.

The getTime and setTime methods are useful for comparing dates. The getTime method returns the number of milliseconds since the epoch for a Date object.

For example, the following code displays the number of days left in the current year:

```
today = new Date()
endYear = new Date("December 31, 1990") // Set day and month
endYear.setYear(today.getYear()) // Set year to this year
msPerDay = 24 * 60 * 60 * 1000 // Number of milliseconds per day
daysLeft = (endYear.getTime() - today.getTime()) / msPerDay
daysLeft = Math.round(daysLeft)
document.write("Number of days left in the year: " + daysLeft)
```

This example creates a Date object named today that contains today's date. It then creates a Date object named endYear and sets the year to the current year. Then, using the number of milliseconds per day, it computes the number of days between today and endYear, using getTime and rounding to a whole number of days.

The parse method is useful for assigning values from date strings to existing Date objects. For example, the following code uses parse and setTime to assign a date value to the IPOdate object:

```
IPOdate = new Date()
IPOdate.setTime(Date.parse("Aug 9, 1995"))
```

Using the Date object: an example

The following example shows a simple application of Date: it displays a continuously-updated digital clock in an HTML text field. This is possible because you can dynamically change the contents of a text field with JavaScript (in contrast to ordinary text, which you cannot update without reloading the document).

The display in Navigator looks like this:

The current time is 6:15:46 P.M.

The <body>: of the document is:

```
<body onLoad="JSClock()">
  <form name="clockForm">
    The current time is
    <input type="text" name="digits" size="12" value="" />
  </form>
```

Note:

- The value
  a number b
- This meth
  conjunctio

getDay()

Returns the
Date object (fr
Note:

- The value
  a number b
  is 0, Monda
- This meth
  conjunctio

getMonth()

Returns th
object (from 0-
Note:

- The value
  is a numb
  January is
  on.
- This meth
  conjunctio

getFullYear()

Returns the
number, from a
Note:

- This meth
  conjunctio

getHours()

Returns the
(from 0-23)
Note:

- The value
  is a two
  the return
  digits, if
  10 it only
- This meth
  conjunctio

getMinutes()

Returns th
object (from 0-
Note:

- The value
  is a two
  the return
  digits, if
  10 it only
- This meth
  conjunctio

getSeconds()

Returns th
object (from 0-
Note:

- The value
  is a two
  the return
  digits, if
  10 it only
- This meth
  conjunctio

```
</body>
```

The <body> tag includes an onLoad event handler. When the page loads, the event handler calls the function JSClock, defined in the <head>. A form called clockForm includes a single text field named digits, whose value is initially an empty string.

The <head> of the document defines JSClock as follows:

```
<head>
  <script type="text/javascript">
    <!--
    function JSClock() {
      var time = new Date()
      var hour = time.getHours()
      var minute = time.getMinutes()
      var second = time.getSeconds()
      var temp = "" + ((hour > 12) ? hour - 12 : hour)
      temp += ((minute < 10) ? ":0" : ":") + minute
      temp += ((second < 10) ? ":0" : ":") + second
      temp += (hour >= 12) ? " P.M." : " A.M."
      document.clockForm.digits.value = temp
      id = setTimeout("JSClock()",1000)
    }
    //-->
  </script>
</head>
```

The JSClock function first creates a new Date object called time; since no arguments are given, time is created with the current date and time. Then calls to the getHours, getMinutes, and getSeconds methods assign the value of the current hour, minute and seconds to hour, minute, and second.

The next four statements build a string value based on the time. The first statement creates a variable temp, assigning it a value using a conditional expression; if hour is greater than 12, (hour - 13), otherwise simply hour.

The next statement appends a minute value to temp. If the value of minute is less than 10, the conditional expression adds a string with a preceding zero; otherwise it adds a string with a demarcating colon. Then a statement appends a seconds value to temp in the same way.

Finally, a conditional expression appends "PM" to temp if hour is 12 or greater; otherwise, it appends "AM" to temp.

The next statement assigns the value of temp to the text field:

```
document.aform.digits.value = temp
```

This displays the time string in the document.

The final statement in the function is a recursive call to JSClock:

```
id = setTimeout("JSClock()", 1000)
```

The built-in JavaScript setTimeout function specifies a time delay to evaluate an expression, in this case a call to JSClock. The second argument indicates a a delay of 1,000 milliseconds (one second). This updates the display of time in the form at one-second intervals.

Note that the function returns a value (assigned to id), used only as an identifier (which can be used by the clearTimeout method to cancel the evaluation).

Manipulate Dates

We can easily manipulate the date by using the methods available for the Date object.

In the example below we set a Date object to a specific date (14th January 2010):

```
var myDate=new Date()
myDate.setFullYear(2010,0,14)
```

And in the following example we set a Date object to be 5 days into the future:

```
var myDate=new Date()
myDate.setDate(myDate.getDate()+5)
```

Note:

If adding five days to a date shifts the month or year, the changes are handled automatically by the Date object itself!

Comparing Dates

The Date object is also used to compare two dates.

The following example compares today's date with the 14th January 2010:

```
var myDate=new Date()
myDate.setFullYear(2010,0,14)
var today = new Date()
if (myDate>today)
  alert("Today is before 14th January 2010")
```

getMilliseconds()	Returns the object (from 0-9 Note: • The va getMillise number. Ho is not alw value is returns tw value is returns on • This meth conjunctio
getTime()	Returns the since midnight , Note: • This meth conjunctio
getTimezoneOffset()	Returns the between local t Time (GMT) Note: • The return is not a c practice o Time. • This meth conjunctio
getUTCDate()	Returns the Date object acc (from 1-31) Note: • The value is a numbe • This meth conjunctio Tip: The Univ (UTC) is the ti Standard.
getUTCDay()	Returns the Date object acc (from 0-6) Note: • The value is a numb Sunday is on. • This meth conjunctio Tip: The Univ (UTC) is the ti Standard.
getUTCMonth()	Returns th object accordi (from 0-11) Note: • The va getUTCMontl 0 and 11. is 1 and s • This meth conjunctio Tip: The Univ (UTC) is the ti

```
else
  alert("Today is after 14th January 2010")
```

Standard.

getUTCFullYear()	Returns the Date object acc Note: • This meth conjunctio Tip: The Univ (UTC) is the ti Standard.
getUTCHours()	Returns the according to un Note: • The va getUTCHour number. Ho is not alw value is returns on • This meth conjunctio Tip: The Univ (UTC) is the ti Standard.
getUTCMinutes()	Returns th object accordi (from 0-59) Note: • The va getUTCMinu number. Ho is not alw value is returns on • This meth conjunctio Tip: The Univ (UTC) is the ti Standard.
getUTCSeconds()	Returns th object accordi (from 0-59) Note: • The va getUTCSeco number. Ho is not alw value is returns on • This meth conjunctio Tip: The Univ (UTC) is the ti Standard.
getUTCMilliseconds()	Returns the object accordi (from 0-999) Note: • The va getUTCMill digit numb value is n if the val only retur the value returns on • This meth conjunctio Tip: The Univ (UTC) is the ti Standard.

parse(datestring)	Takes a dat number of mill of January 1, 1 Argument: • datestring representi
setDate()	Sets the da Arguments: • day (Requi Note: • This metho
setMonth(month,day)	Sets the m (from 0-11) Arguments: • month (Req between 0 month • day (Opti between 1 date Note: • The value number bet is 0, Febr • This meth conjunctio
setFullYear(year,month,day)	Sets the y (four digits) Arguments: • year (Requ representi • month (Opt between 0 month • day (Opti between 1 date Note: This met conjunction wit
setHours(hour,min,sec,millisec)	Sets the ho Argument: • hour (Requ • min (Requi • sec (Optio • millisec (Note: • If one of two leadin • This metho
setMinutes(min,sec,millisec)	Set the min Argument: • min (Requi • sec (Optio • millisec (Note: • If one of

	two leadin • This metho
setSeconds(sec,millisec)	Sets the se Argument: • sec (Requi • millisec (Note: • If one of two leadin • This metho
setMilliseconds(millisec)	Sets the mi Argument: • millisec (Note: • If the par one or two • This metho
setTime(millisec)	Calculates adding or sul number of milli January 1, 1970 Argument: • millisec value millisecon January 1, number Note: This met conjunction wit
setUTCDate(day)	Sets the da Argument: • day (Requi Note: This meth Tip: The Univer
setUTCMonth(month,day)	Sets the m according to un Arguments: • month (Req between 0 month • day (Opti between 1 date Note: This met conjunction wit Tip: The Univ (UTC) is the ti Standard.
setUTCFullYear(year,month,day)	Sets the y according to digits) Arguments: • year (Requ representi • month (Opt between 0 month • day (Opti between 1 date Note: This met conjunction wit Tip: The Univ (UTC) is the ti

	Standard.
setUTCHours (hour,min,sec,millisec)	Sets the hou Arguments: • hour (Requ • min (Optio • sec (Optio • millisec (Note: • If one of two leadin • This metho Tip: The Univer
setUTCMinutes(min,sec,millisec)	Set the min Arguments: • min (Requi • sec (Optio • millisec (Note: • If one of two leadin • This metho Tip: The Univer
setUTCSeconds(sec,millisec)	Set the sec Arguments: • sec (Requi • millisec (Note: • If one of two leadin • This metho Tip: The Univer
setUTCMilliseconds(millisec)	Sets the mi Argument: • millisec (Note • If the par one or two • This metho Tip: The Univer
toUTCString()	Converts a to universal ti
toLocaleString()	Converts a to local time,
UTC (year,month,day, hours,minutes,seconds,ms)	Takes a dat of millisecond January 1, 1970 time Arguments • year (Req number rep • month (R

between 0
month
- day (Requi
1 and 31 r
- hours (0
between 0
hour
- minutes
between 0
minutes
- seconds
between 0
seconds
- ms (Optiona
and 999
millisecon

JavaScript Func

The built-in Function object specifies a string of JavaScript code to be compiled as a function.
To create a Function object:

```
functionObjectName = new Function ([arg1, arg2, ... argn], functionBody)
```

Arguments:

- functionObjectName is the name of a variable or a property of an existing object. It can also be an object followed by a lowe
object or a property of an existing object.
- arg1, arg2, ... argn are arguments to be used by the function as formal argument names. Each must be a string that corresponds
- functionBody is a string specifying the JavaScript code to be compiled as the function body.

Function objects are evaluated each time they are used. This is less efficient than declaring a function and calling it within y
In addition to defining functions as described here, you can also use the function statement, as described in "function".
The following code assigns a function to the variable setBGColor. This function sets the current document's background color.

```
var setBGColor = new Function("document.bgColor='antiquewhite'")
```

To call the Function object, you can specify the variable name as if it were a function. The following code executes the functic

```
var colorChoice="antiquewhite"
if (colorChoice=="antiquewhite") {setBGColor()}
```

You can assign the function to an event handler in either of the following ways:

```
1. document.form1.colorButton.onclick=setBGColor
2. <input name="colorButton" type="button" value="Change background color" onClick="setBGColor()">
```

Creating the variable setBGColor shown above is similar to declaring the following function:

```
function setBGColor() {
  document.bgColor='antiquewhite'
}
```

Assigning a function to a variable is similar to declaring a function, but they have differences:

- When you assign a function to a variable using var setBGColor = new Function("..."), setBGColor is a variable for which the cur
- When you create a function using function setBGColor() {...}, setBGColor is not a variable, it is the name of a function.

JavaScript Image Object Description

To create an Image object:

border

```
imageName = new Image([width, height])
```

To use an Image object's properties:

```
1. imageName.propertyName
2. document.images[index].propertyName
3. formName.elements[index].propertyName
```

To define an event handler for an Image object created with the Image() constructor:

complete

```
1. imageName.onabort = handlerFunction
2. imageName.onerror = handlerFunction
3. imageName.onload = handlerFunction
```

height

The position and size of an image in a document are set when the document is displayed in Navigator and cannot be changed using JavaScript (the width and height properties are read-only). You can change which image is displayed by setting the src and lowsrc properties. (See the descriptions of src and lowsrc.)

hspace

You can use JavaScript to create an animation with an Image object by repeatedly setting the src property. JavaScript animation is slower than GIF animation, because with GIF animation the entire animation is in one file; with JavaScript animation, each frame is in a separate file, and each file must be loaded across the network (host contacted and data transferred).

lowsrc

Image objects do not have onClick, onMouseOut, and onMouseOver event handlers. However, if you define an Area object for the image or place the <img> tag within a Link object, you can use the Area or Link object's event handlers. See the Link object.

The Image() constructor

The primary use for an Image object created with the Image() constructor is to load an image from the network (and decode it) before it is actually needed for display. Then when you need to display the image within an existing image cell, you can set the src property of the displayed image to the same value as that used for the prefetched image, as follows.

```
myImage = new Image()
myImage.src = "seaotter.gif"
...
document.images[0].src = myImage.src
```

The resulting image will be obtained from cache, rather than loaded over the network, assuming that sufficient time has elapsed to load and decode the entire image. You can use this technique to create smooth animations, or you could display one of several images based on form input.

name

src

vspace

width

JavaScript Math Object Description

The built-in Math object has properties and methods for mathematical constants and functions. For example, the Math object's PI property has the value of pi (3.141...), which you would use in an application as

```
Math.PI
```

Similarly, standard mathematical functions are methods of Math. These include trigonometric, logarithmic, exponential, and other functions. For example, if you want to use the trigonometric function sine, you would write

```
Math.sin(1.56)
```

E
LN2
LN10
LOG2E
LOG10E
PI
SQRT1_2
SQRT2

Note:

Trigonometric methods of Math take arguments in radians.

It is often convenient to use the with statement when a section of code uses several math constants and methods, so you don't have to type "Math" repeatedly. For example,

```
with (Math) {
  a = PI*r*r
  y = r*sin(theta)
  x = r*cos(theta)
}
```

abs(x)
acos(x)
asin(x)
atan(x)
atan2(y,x)
ceil(x)
cos(x)
exp(x)
floor(x)
log(x)
max(x,y)
min(x,y)
pow(x,y)
random()
round(x)
sin(x)
sqrt(x)
tan(x)

JavaScript Number Object Description

The Number object has properties for numerical constants, such as maximum value, not-a-number, and infinity. You use these properties as follows:

```
biggestNum = Number.MAX_VALUE
smallestNum = Number.MIN_VALUE
infiniteNum = Number.POSITIVE_INFINITY
negInfiniteNum = Number.NEGATIVE_INFINITY
notANum = Number.NaN
```

MAX_VALUE
MIN_VALUE
NaN
NEGATIVE_INFIN
POSITIVE_INFIN

JavaScript String Object Description

JavaScript does not have a string data type. However, you can use the String object and its methods to work with strings in your applications. The String object has a large number of methods for manipulating strings. It has one property for determining the string's length.

To create a String object:

```
stringObjectName = new String(string)
```

Arguments:

- stringObjectName is the name of a new String object.
- string is any string.

For example, the following statement creates a String object called mystring:

```
mystring = new String ("Hello, World!")
```

length	Returns the
anchor(anchorname)	Creates an I Argument: • anchorname name for tl
big()	Displays a

String literals are also String objects; for example, the literal "Howdy" is a String object.

A String object has one property, length, that indicates the number of characters in the string. So, using the previous example, the expression

```
x = mystring.length
```

assigns a value of 13 to x, because "Hello, World!" has 13 characters.

A String object has two types of methods: those that return a variation on the string itself, such as substring and toUpperCase, and those that return an HTML-formatted version of the string, such as bold and link.

For example, using the previous example, both mystring.toUpperCase() and "hello, world!".toUpperCase() return the string "HELLO, WORLD!".

The substring method takes two arguments and returns a subset of the string between the two arguments. Using the previous example, mystring.substring(4, 9) returns the string "o, Wo." For more information, see the reference topic for substring.

The String object also has a number of methods for automatic HTML formatting, such as bold to create boldface text and link to create a hyperlink. For example, you could create a hyperlink to a hypothetical URL with the link method as follows:

```
mystring.link("http://www.helloworld.com")
```

blink()	Displays a l Note: This Internet Explor
bold()	Displays a
charAt(index)	Returns t specified posit Argument: • index (representi string Note: The fi string is at po
charCodeAt(index)	Returns t character at a Argument: • index (representi string Note: The fi string is at po
concat(stringX,stringX,...,stringX)	Joins two o Argument: • stringX (string obj string
fixed()	Displays a
fontcolor(color)	Displays a color Argument: • color(Requ color for can be a c value (rg number (#F
fontsize(size)	Displays a size Argument: • size(Requi specifies Note: The size number from 1 t
fromCharCode(numX,numX,...,numX)	Takes the s and returns a s Argument: • numX(Requi values Note: This met of String - it of a String created. The String.fromChar myStringObject.
indexOf(searchvalue,fromindex)	Returns the occurrence of a in a string Arguments: • searchvalu string val • fromindex(where to s Notes:

	• The index sensitive! • This meth string val occurs.
italics()	Displays a s
lastIndexOf(searchvalue,fromindex)	Returns the occurrence of a searching backw position in a s Arguments: • search(Requ string valu • fromindex(where to Starting ba Notes: • The index sensitive! • This meth string val occurs.
link()	Displays a s
match(searchvalue)	Searches fo value in a strin This method and lastIndexOf specified stri position of the Argument: • searchvalue string valu Notes: • The matc sensitive! • This metho string val occurs.
replace(findstring,newstring)	Replaces so other character Arguments: • findstring Specifies To perform 'g' flag t perform a add an 'i' • newstring(string to from finds Note: • The repla sensitive.
search(searchstring)	Searches a value Argument: • searchstrin The value string. insensitive Notes:

	• The searc sensitive. • The search position o the string it returns
slice(start,end)	Extracts a returns the ex string Argument: • start(Requ start the number • end(Optiona the select Notes: • You can us to select string. • If end is selects al specified the end of
small()	Displays a
split(separator, howmany)	Splits a s strings Arguments: • separator(character, substring determine string • howmany(Op times spli a numeric Note: • If an empt the separa between ea
strike()	Displays strikethrough
sub()	Displays a
substr(start,length)	Extracts a characters in a index Arguments: • start(Requ extraction. value • length(Opt characters numeric va Notes: • To extract of the s start numb • The start • If the omitted, t

	the end of
substring(start,stop)	Extracts th between two spe Arguments: • start(Requ extraction. value • stop(Optio extraction. value Notes: • To extract of the s start numb • The start • If the sto this metho the string.
sup()	Displays a
toLowerCase()	Displays a letters
toUpperCase()	Displays a letters

JavaScript Event

onabort	Loading of an image is interrupted
onblur	An element loses focus
onchange	The content of a field changes
onclick	Mouse clicks an object
ondblclick	Mouse double-clicks an object
onerror	An error occurs when loading a document or an image
onfocus	An element gets focus
onkeydown	A keyboard key is pressed
onkeypress	A keyboard key is pressed or held down
onkeyup	A keyboard key is released
onload	A page or an image is finished loading
onmousedown	A mouse button is pressed
onmousemove	The mouse is moved
onmouseout	The mouse is moved off an element
onmouseover	The mouse is moved over an element
onmouseup	A mouse button is released
onreset	The reset button is clicked
onresize	A window or frame is resized
onselect	Text is selected
onsubmit	The submit button is clicked
onunload	The user exits the page