JavaScript

Introduction

Java script is one of the scripting languages which facilitate a disciplined approach to design computer programs that enhance the functionality and appearance of web pages.

Example:

```
<script language = "javascript">
        document.writeln("<h1>Welcome to JavaScript Programming!</h1>" );
        document.writeln( "<h1>Welcome to<br />JavaScript" +
        "<br />Programming!</h1>" );
        document.write( "<h1 style = \"color: blue\">" );
        document.write( "Welcome to JavaScript Programming!</h1>" );
</script>
```

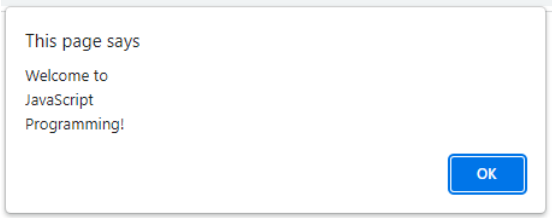
Welcome to JavaScript Programming!

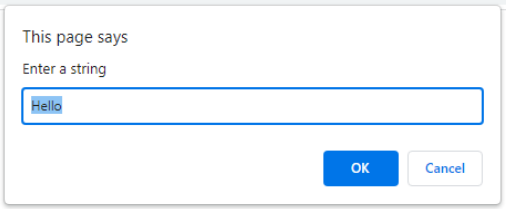
Welcome to
JavaScript
Programming!

Welcome to JavaScript Programming!

Alert, Prompt, and Confirm

```
<script language = "javascript">
window.alert( "Welcome to\nJavaScript\nProgramming!" );
var s;
s = window.prompt( "Enter a string","Hello" );
document.writeln("<h1>You have entered " + s + "</h1>");
</script>
```





You have entered Java Script

Variables and Arithmetic Expressions

- Arithmetic: +, -, *, /, %
- Logical: <, >, <=, >=, ==,!=
- Additional "shorthand" assignment; operations like

  count +=; // -=, *=, /=, %=, etc.
- Incrementing or decrementing by 1, shorten further to

  count++; count--;

Example:

```
<script language = "javascript">
var firstNumber, secondNumber, number1, number2, sum;
firstNumber = window.prompt( "Enter first integer", "0" );
secondNumber = window.prompt( "Enter second integer", "0" );
number1 = parseInt( firstNumber );
number2 = parseInt( secondNumber );
sum = number1 + number2;
document.writeln( "<h1>The sum is " + sum + "</h1>" );
</script>
```

Control Structures

- if (condition)

  { Statements 1; }

  else

  { Statements 2; }
- switch(exp)

{

  case const1: statements 1;

  case const2: statements 2;

  case const3: statements 3;

```
        default : statements_default;
}
```

Example:

```
<SCRIPT LANGUAGE="JavaScript">
order = prompt( "Total order? ", 0 ); // get amount spent
if ( order < 100.0 ) {
        discount = order * 0.1;
}
else if ( order < 500.0 ) {
        discount = order * 0.15;
}
else {
        discount = order * 0.2;
}
document.write( "Discount = $", discount, "<BR>" );
</SCRIPT>
```

```
while (condition)
{
Statements to be repeated;
}
```

```
for (exp1;exp2;exp3)
{
Statements to be repeated;
}
```

```
do
{
Statements to be repeated;
}
while (condition);
```

Example:

```
<table width=25% border=3>
<caption>Ten Numbers With Their Squares</caption>
<tr><td>Number</td><td>Square</td></tr>
```

```
<SCRIPT Language="JavaScript">
var count;
for (count=1 ; count <=10 ; count++)
{
        document.writeln("<tr>");
        document.writeln("<td align=center>" + count + "</td>");
        document.writeln("<td align=center>" + count*count + "</td>");
        document.writeln("</tr>");
}
</SCRIPT></table>
```

Functions

function *function_name* (parameter_1, parameter_2, … , parameter_n)

{

statements of the function;

it may contain return statement;

}

Functions can be classified into to two types; function with return value and function without return value.

var = function_name(argument_1, argument_2, ... , argument_n);

function_name(argument_1, argument_2, … , argument_n);

Example:

```
<table width=25% border=3>
<caption>Ten Numbers With Their Squares</caption>
<tr><td>Number</td><td>Square</td></tr>
<SCRIPT Language="JavaScript">
var count;
for (count=1 ; count <=10 ; count++)
```

```
{
display("<tr>");
display("<td align=center>" + count + "</td>");
display("<td align=center>" + square(count) + "</td>");
display("</tr>");
}
function square(x)
{
return x*x;
}
function display(s)
{
document.writeln(s);
}
</SCRIPT></table>
```

Example:

```
<title>Finding the Maximum of Three Values</title>
<SCRIPT Language="JavaScript">
var input1 = window.prompt( "Enter first number", "0" );
var input2 = window.prompt( "Enter second number", "0" );
var input3 = window.prompt( "Enter third number", "0" );
var value1 = parseFloat( input1 );
var value2 = parseFloat( input2 );
var value3 = parseFloat( input3 );
var maxValue = maximum( value1, value2, value3 );
document.writeln( "First number: " + value1 + "<br />Second number: "+value2
+ "<br />Third number: " + value3 + "<br />Maximum is: " + maxValue );
```

```
// maximum method definition (called from line 25)
function maximum( x, y, z )
{
return Math.max( x, Math.max( y, z ) );
}
</script>
```

Arrays

var *array_name* = new Array(*number_of_elements*);

e.g. var c = new Array(5);

create an array with initial values as follows:

var *array_name* = [val_1, val_2, …. , val_3];

var *array_name* = new Array(val_1, val_2, …. , val_3);

e.g. var x = [3,5,8,2,9];

Example:

```
<title>Initializing an Array</title>
<SCRIPT Language="JavaScript">
var array1 = [ 1, 2, 3 ];
var array2 = [ 50,30,99,32,52 ] ;
outputArray ( "Values in array1", array1 );
outputArray ( "Values in array2", array2 );

var n1 = new Array( 5 ); // allocate 5-element Array
var n2 = new Array(); // allocate empty Array
for ( var i = 0; i < n1.length; ++i )
        n1[ i ] = i;
for ( i = 0; i < 5; ++i )
        n2[ i ] = i;
```

```
outputArray( "Array n1 contains", n1 );
outputArray( "Array n2 contains", n2 );
function outputArray( header, theArray )
{
document.writeln( "<h2>" + header + "</h2>" );
document.writeln( "<table border = \"1\" width =" + "\"50%\">" );
document.writeln( "<tr><td align = left width=50%>Subscript</td>");
document.writeln( "<td align=left>Value</td></tr><tr>" );
for ( var i = 0; i < theArray.length; i++ )
document.writeln( "<tr><td>" + i + "</td><td>" + theArray[ i ] +"</td></tr>" );
document.writeln( "</tr></table>" );
}
</script></head>
```

Two dimensions Arrays

- var *array_name* = new Array(*number_of_rows*);
- *array_name*[0] = new Array(*number_of_elements_in_the_first_row*);
- *array_name*[1] = new Array(*number_of_elements_in_the_second_row*);
- and so on.
- e.g. var c = new Array(3);

  c[0] = new Array(5);

  c[1] = new Array(2);

  c[2] = new Array(7);

  var x = [[1,2] , [3,4,5]];

Example:

```
<title>Initializing Multidimensional Arrays</title>
<SCRIPT Language="JavaScript">
var array1 = [ [ 1, 2, 3 ], // first row
```

```
[ 4, 5, 6 ] ]; // second row
var array2 = [ [ 1, 2 ], // first row
[ 3 ], // second row
[ 4, 5, 6 ] ]; // third row
outputArray( "Values in array1 by row", array1 );
outputArray( "Values in array2 by row", array2 );
function outputArray( header, theArray )
{
document.writeln( "<h2>" + header + "</h2><tt>" );
for ( var i in theArray )
{
for ( var j in theArray[ i ] )
document.write( theArray[ i ][ j ] + " " );
document.writeln( "<br />" );
}
document.writeln( "</tt>" );
}
</script>
```

Math Object

Method	Description	Example
abs(x)	absolute value of **x**	**abs(** 7.2 **)** is 7.2 **abs(** 0.0 **)** is 0.0 **abs(** -5.6 **)** is 5.6
ceil(x)	rounds **x** to the smallest integer not less than **x**	**ceil(** 9.2 **)** is 10.0 **ceil(** -9.8 **)** is -9.0
cos(x)	trigonometric cosine of **x** (**x** in radians)	**cos(** 0.0 **)** is 1.0
exp(x)	exponential method **ex**	**exp(** 1.0 **)** is 2.71828 **exp(** 2.0 **)** is 7.38906
floor(x)	rounds **x** to the largest integer not greater than **x**	**floor(** 9.2 **)** is 9.0 **floor(** -9.8 **)** is -10.0
log(x)	natural logarithm of **x** (base *e*)	**log(** 2.718282 **)** is 1.0 **log(** 7.389056 **)** is 2.0
max(x, y)	larger value of **x** and **y**	**max(** 2.3, 12.7 **)** is 12.7 **max(** -2.3, -12.7 **)** is -2.3
min(x, y)	smaller value of **x** and **y**	**min(** 2.3, 12.7 **)** is 2.3 **min(** -2.3, -12.7 **)** is -12.7
pow(x, y)	**x** raised to power **y** (xy)	**pow(** 2.0, 7.0 **)** is 128.0 **pow(** 9.0, .5 **)** is 3.0
round(x)	rounds **x** to the closest integer	**round(** 9.75 **)** is 10 **round(** 9.25 **)** is 9
sin(x)	trigonometric sine of **x** (**x** in radians)	**sin(** 0.0 **)** is 0.0
sqrt(x)	square root of **x**	**sqrt(** 900.0 **)** is 30.0 **sqrt(** 9.0 **)** is 3.0
tan(x)	trigonometric tangent of **x** (**x** in radians)	**tan(** 0.0 **)** is 0.0

String Object

Method	Description
charAt(***index*** **)**	Returns a string containing the character at the specified *index*. If there is no character at that *index*, **charAt** returns an empty string. The first character is located at *index* 0.
charCodeAt(***index*** **)**	Returns the Unicode value of the character at the specified *index*. If there is no character at that *index*, **charCodeAt** returns **NaN**.
concat(***string*** **)**	Concatenates its argument to the end of the string that invokes the method. The string invoking this method is not modified; rather a new **String** is returned. This method is the same as adding two strings with the string concatenation operator + (e.g., **s1.concat(s2)** is the same as **s1** + **s2**).
fromCharCode(***value1, value2,*** **)**	Converts a list of Unicode values into a string containing the corresponding characters.
indexOf(*substring, index*)	Searches for the first occurrence of *substring* starting from position *index* in the string that invokes the method. The method returns the starting index of *substring* in the source string or –1 if *substring* is not found. If the *index* argument is not provided, the method begins searching from index 0 in the source string.
lastIndexOf(***substring, index*** **)**	Searches for the last occurrence of *substring* starting from position *index* and searching toward the beginning of the string that invokes the method. The method returns the starting index of *substring* in the source string or –1 if *substring* is not found. If the *index* argument is not provided, the method begins searching from end of the source string.
slice(***start, end*** **)**	Returns a string containing the portion of the string from index *start* through index *end*. If the *end* index is not specified, the method returns a string from the *start* index to the end of the source string. A negative *end* index specifies an offset from the end of the string starting from a position one past the end of the last character (so, –1 indicates the last character position in the string).

split(*string*)	Splits the source string into an array of strings (tokens) where its *string* argument specifies the delimiter (i.e., the characters that indicate the end of each token in the source string).
substr(*start*, *length*)	Returns a string containing *length* characters starting from index *start* in the source string. If *length* is not specified, a string containing characters from *start* to the end of the source string is returned.
substring(*start*, *end*)	Returns a string containing the characters from index *start* up to but not including index *end* in the source string.
toLowerCase()	Returns a string in which all uppercase letters are converted to lowercase letters. Non-letter characters are not changed.
toUpperCase()	Returns a string in which all lowercase letters are converted to uppercase letters. Non-letter characters are not changed.
toString()	Returns the same string as the source string.
valueOf()	Returns the same string as the source string.

Example:

```
<title>Character Processing Methods</title>
<SCRIPT Language="JavaScript">
var s = "ZEBRA";
var s2 = "AbCdEfG";
document.writeln( "<p>Character at index 0 in '" +s + "' is " + s.charAt( 0 ) );
document.writeln( "<br />Character code at index 0 in '"+ s + "' is " + s.charCodeAt( 0 ) +
"</p>" );
document.writeln( "<p>'" + String.fromCharCode( 87, 79, 82, 68 ) +"' contains character codes
87, 79, 82 and 68</p>" )
document.writeln( "<p>'" + s2 + "' in lowercase is '" +s2.toLowerCase() + "'" );
document.writeln( "<br />'" + s2 + "' in uppercase is '"+ s2.toUpperCase() + "'</p>" );
</script>
```

Example:

```
<title>String Method split and substring</title>
<SCRIPT Language="JavaScript">
var str = prompt("Enter a String");
var newstr,s;
s = str.split(" ");
newstr = s.join( "\n" );
document.writeln("<pre>" + newstr + "</pre>");
</script>
```

Date Object

Method	Description
getDate() **getUTCDate()**	Returns a number from 1 to 31 representing the day of the month in local time or UTC, respectively.
getDay() **getUTCDay()**	Returns a number from 0 (Sunday) to 6 (Saturday) representing the day of the week in local time or UTC, respectively.
getFullYear() **getUTCFullYear()**	Returns the year as a four-digit number in local time or UTC, respectively.
getHours() **getUTCHours()**	Returns a number from 0 to 23 representing hours since midnight in local time or UTC, respectively.
getMilliseconds() **getUTCMilliSeconds()**	Returns a number from 0 to 999 representing the number of milliseconds in local time or UTC, respectively. The time is stored in hours, minutes, seconds and milliseconds.
getMinutes() **getUTCMinutes()**	Returns a number from 0 to 59 representing the minutes for the time in local time or UTC, respectively.
getMonth() **getUTCMonth()**	Returns a number from 0 (January) to 11 (December) representing the month in local time or UTC, respectively.

getSeconds() **getUTCSeconds()**	Returns a number from 0 to 59 representing the seconds for the time in local time or UTC, respectively.
getTime()	Returns the number of milliseconds between January 1, 1970 and the time in the **Date** object.
getTimezoneOffset()	Returns the difference in minutes between the current time on the local computer and UTC—previously known as Greenwich Mean Time (GMT).
setDate(*val* **)** **setUTCDate(** *val* **)**	Sets the day of the month (1 to 31) in local time or UTC, respectively.
setFullYear(*y*, *m*, *d* **)** **setUTCFullYear(** *y*, *m*, *d* **)**	Sets the year in local time or UTC, respectively. The second and third arguments representing the month and the date are optional. If an optional argument is not specified, the current value in the **Date** object is used.
setHours(*h*, *m*, *s*, *ms* **)** **setUTCHours(** *h*, *m*, *s*, *ms* **)**	Sets the hour in local time or UTC, respectively. The second, third and fourth arguments representing the minutes, seconds and milliseconds are optional. If an optional argument is not specified, the current value in the **Date** object is used.
setMilliSeconds(*ms* **)** **setUTCMilliseconds(** *ms* **)**	Sets the number of milliseconds in local time or UTC, respectively.

setMilliSeconds(*ms* **)** **setUTCMilliseconds(** *ms* **)**	Sets the number of milliseconds in local time or UTC, respectively.
setMinutes(*m*, *s*, *ms* **)** **setUTCMinutes(** *m*, *s*, *ms* **)**	Sets the minute in local time or UTC, respectively. The second and third arguments representing the seconds and milliseconds are optional.
setMonth(*m*, *d* **)** **setUTCMonth(** *m*, *d* **)**	Sets the month in local time or UTC, respectively. The second argument representing the date is optional. If the optional argument is not specified, the current date value in the **Date** object is used.
setSeconds(*s*, *ms* **)** **setUTCSeconds(** *s*, *ms* **)**	Sets the second in local time or UTC, respectively. The second argument representing the milliseconds is optional. If this argument is not specified, the current millisecond value in the **Date** object is used.
setTime(*ms* **)**	Sets the time based on its argument—the number of elapsed milliseconds since January 1, 1970.
toLocaleString()	Returns a string representation of the date and time in a form specific to the computer's locale. For example, September 13, 2001 at 3:42:22 PM is represented as *09/13/01 15:47:22* in the United States and *13/09/01 15:47:22* in Europe.
toUTCString()	Returns a string representation of the date and time in the form: *19 Sep 2001 15:47:22 UTC*

toString()	Returns a string representation of the date and time in a form specific to the locale of the computer (*Mon Sep 19 15:47:22 EDT 2001* in the United States).
valueOf()	The time in number of milliseconds since midnight, January 1, 1970.

Example:

```
<title>Date and Time Methods</title>
<SCRIPT Language="JavaScript">
var current = new Date();
document.writeln( "<h1>String representations and valueOf</h1>" );
document.writeln( "toString: " + current.toString() +
"<br />toLocaleString: " + current.toLocaleString() +
"<br />toUTCString: " + current.toUTCString() +
"<br />valueOf: " + current.valueOf() );
document.writeln( "<h1>Get methods for local time zone</h1>" );
document.writeln( "getDate: " + current.getDate() +
"<br />getDay: " + current.getDay() +
"<br />getMonth: " + current.getMonth() +
"<br />getFullYear: " + current.getFullYear() +
"<br />getTime: " + current.getTime() +
"<br />getHours: " + current.getHours() +
"<br />getMinutes: " + current.getMinutes() +
"<br />getSeconds: " + current.getSeconds() +
"<br />getMilliseconds: " +
current.getMilliseconds() +
"<br />getTimezoneOffset: " +
current.getTimezoneOffset() );
document.writeln("<h1>Specifying arguments for a new Date</h1>" );
```

```
var anotherDate = new Date( 2001, 2, 18, 1, 5, 0, 0 );
document.writeln( "Date: " + anotherDate );
document.writeln("<h1>Set methods for local time zone</h1>" );
anotherDate.setDate( 31 );
anotherDate.setMonth( 11 );
anotherDate.setFullYear( 2001 );
anotherDate.setHours( 23 );
anotherDate.setMinutes( 59 );
anotherDate.setSeconds( 59 );
document.writeln( "Modified date: " + anotherDate );
</script>
```