**J1**

**JavaScript is the most popular programming language in the world.  
  
It is the language for HTML, for the Web, for computers, servers, laptops, tablets, smart phones, and more.  
  
This page contains some examples of what JavaScript can do in HTML.  
  
JavaScript Can Change HTML Elements  
The HTML DOM (the Document Object Model) is the official W3C standard for accessing HTML elements.  
  
JavaScript can manipulate the DOM (change HTML contents).  
  
The following example changes the content (innerHTML) of an HTML element identified with id="demo":  
  
Example  
  
document.getElementById("demo").innerHTML = "Hello JavaScript";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My First JavaScript</h1>  
  
< p>JavaScript can change the content of an HTML element:</p>  
  
< button type="button" onclick="myFunction()">Click Me!</button>  
  
< p id="demo">This is a demonstration.</p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = "Hello JavaScript!";  
}  
< /script>  
  
< /body>  
< /html>  
  
The method document.getElementById() is one of many methods in the HTML DOM.  
  
You can use JavaScript to:  
  
Change HTML elements  
Delete HTML elements  
Create new HTML elements  
Copy and clone HTML elements  
And much more ...  
There are several chapters, about the HTML DOM, later in this tutorial.  
  
JavaScript Can Change HTML Attributes  
This example changes the value of the source attribute (src) of an HTML <image> element:  
  
The Light bulb  
  
  
Click the light bulb to turn on/off the light  
  
< !DOCTYPE html>  
< html>  
< body>  
< script>  
function changeImage() {  
    var image = document.getElementById('myImage');  
    if (image.src.match("bulbon")) {  
        image.src = "pic\_bulboff.gif";  
    } else {  
        image.src = "pic\_bulbon.gif";  
    }  
}  
< /script>  
  
< img id="myImage" onclick="changeImage()" src="pic\_bulboff.gif" width="100" height="180">  
  
< p>Click the light bulb to turn on/off the light</p>  
  
< /body>  
< /html>  
  
  
With JavaScript, you can change almost any HTML attribute.  
  
JavaScript Can Change HTML Styles (CSS)  
Changing the style of an HTML element, is a variant of changing an HTML attribute.  
  
Example  
  
document.getElementById("demo").style.fontSize = "25px";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My First JavaScript</h1>  
  
< p id="demo">JavaScript can change the style of an HTML element.</p>  
  
< script>  
function myFunction() {  
    var x = document.getElementById("demo");  
    x.style.fontSize = "25px";  
    x.style.color = "red";  
}  
< /script>  
  
< button type="button" onclick="myFunction()">Click Me!</button>  
  
< /body>  
< /html>  
  
With JavaScript, you can change almost any CSS values.  
  
JavaScript Can Validate Data  
JavaScript is often used to validate input:  
  
Please input a number between 1 and 10  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Please input a number between 1 and 10:</p>  
  
< input id="numb" type="text">  
  
< button type="button" onclick="myFunction()">Submit</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var x, text;  
  
    //Get the value of input field with id="numb"  
  
    x = document.getElementById("numb").value;  
  
    // If x is Not a Number or less than one or greater than 10  
  
    if (isNaN(x) || x < 1 || x > 10) {  
        text = "Input not valid";  
    } else {  
        text = "Input OK";  
    }  
    document.getElementById("demo").innerHTML = text;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Did You Know?  
Note    JavaScript and Java are two completely different languages, both in concept and design.  
  
JavaScript was invented by Brendan Eich, to be used in Netscape (a no longer existing browser) in 1995,  
and was adopted by the ECMA standard association in 1997.  
  
ECMA-262 is the official name. ECMAScript 5 (version 1.8.5 - July 2010) is the latest standard.**

**J2**

**In HTML, JavaScripts must be inserted between <script> and </script> tags.  
  
JavaScripts can be put in the <body> and in the <head> section of an HTML page.  
  
The <script> Tag  
To insert a JavaScript into an HTML page, use the <script> tag.  
  
The <script> and </script> tells where the JavaScript starts and ends.  
  
The lines between <script> and </script> contain the JavaScript code:  
  
Example  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = "My First JavaScript Function";  
}  
< /script>  
You don't have to understand the code above.  
  
Just take it for a fact, that the browser will interpret the code between the <script> and </script> tags as JavaScript.  
  
Note    Old examples may have type="text/javascript" in the <script> tag. This is no longer required.  
JavaScript is the default scripting language in all modern browsers and in HTML5.  
  
JavaScript Functions and Events  
Often, JavaScript code is written to be executed when an event occurs, like when the user clicks a button.  
  
JavaScript code inside a function, can be invoked later, when an event occurs.  
  
Invoke a function = Call upon a function (ask for the code in the function to be executed).  
  
You will learn much more about functions and events in later chapters.  
  
JavaScript in <head> or <body>  
You can place any number of scripts in an HTML document.  
  
Scripts can be placed in the <body> or in the <head> section of HTML, and/or in both.  
  
Often you will see scripts at the bottom of the <body> section of a web page. This can reduce display time.  
  
Sometimes you will see all JavaScript functions in the <head> section.  
  
Anyway, separating HTML and JavaScript, by putting all the code in one place, is always a good habit.  
  
JavaScript in <head>  
In this example, a JavaScript function is placed in the <head> section of an HTML page.  
  
The function is invoked (called) when a button is clicked:  
  
Example  
  
< !DOCTYPE html>  
< html>  
< head>  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = "Paragraph changed.";  
}  
< /script>  
< /head>  
  
< body>  
  
< h1>My Web Page</h1>  
  
< p id="demo">A Paragraph</p>  
  
< button type="button" onclick="myFunction()">Try it</button>  
  
< /body>  
< /html>  
  
  
JavaScript in <body>  
In this example, a JavaScript function is placed in the <body> section of an HTML page.  
  
The function is invoked (called) when a button is clicked:  
  
Example  
  
< !DOCTYPE html>  
< html>  
  
< body> <h1>My Web Page</h1>  
  
< p id="demo">A Paragraph</p>  
  
< button type="button" onclick="myFunction()">Try it</button>  
  
< script>  
function myFunction() {  
   document.getElementById("demo").innerHTML = "Paragraph changed.";  
}  
< /script>  
  
< /body>  
< /html>  
  
Note    It is a good idea to place scripts at the bottom of the <body> element.  
This improves page load, because HTML loading is not blocked by scripts loading.  
External JavaScripts  
Scripts can also be placed in external files.  
  
External scripts are practical when the same code is used in many different web pages.  
  
JavaScript files have the file extension .js.  
  
To use an external script, put the name of the script file in the source (src) attribute of the <script> tag:  
  
Example  
  
< !DOCTYPE html>  
< html>  
< body>  
< script src="myScript.js"></script>  
< /body>  
< /html>  
  
  
You can place an external script reference in <head> or <body> as you like.  
  
The script will behave as if it was located exactly where you put the reference in the HTML document.  
Note    External scripts cannot contain <script> tags.**

**J3**

**JavaScript does not have any print or output functions.  
  
In HTML, JavaScript can only be used to manipulate HTML elements.  
  
Manipulating HTML Elements  
To access an HTML element from JavaScript, you can use the document.getElementById(id) method.  
  
Use the "id" attribute to identify the HTML element, and innerHTML to refer to the element content:  
  
Example  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My First Web Page</h1>  
  
< p id="demo">My First Paragraph</p>  
  
< script>  
document.getElementById("demo").innerHTML = "Paragraph changed.";  
< /script>  
  
< /body>  
< /html>  
  
  
The JavaScript statement above (inside the <script> tag) is executed by the web browser:  
  
document.getElementById("demo") is JavaScript code for finding an HTML element using the id attribute.  
  
innerHTML = "Paragraph changed." is JavaScript code for changing an element's HTML content (innerHTML).  
  
In This Tutorial  
Most of the time, in this tutorial, we will use the output method described above:  
  
Writing output into a <p> element with id="demo".  
  
Writing to The HTML Document  
For testing purposes, you can use JavaScript to write directly to the HTML document:  
  
Example  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My First Web Page</h1>  
  
< p>My first paragraph.</p>  
  
< script>  
document.write(Date());  
< /script>  
  
< /body>  
< /html>  
  
Note  
Use document.write for testing only.  
If you execute it, on a loaded HTML document, all HTML elements will be overwritten.  
  
Example  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My First Web Page</h1>  
  
< p>My first paragraph.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< script>  
function myFunction() {  
       document.write(Date());  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Writing to The Console  
If your browser supports debugging, you can use the console.log() method to display JavaScript values in the browser.  
  
Activate debugging in your browser with F12, and select "Console" in the debugger menu.  
  
Example  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My First Web Page</h1>  
  
< script>  
a = 5;  
b = 6;  
c = a + b;  
console.log(c);  
< /script>  
  
< /body>  
< /html>  
  
  
Did You Know?  
Note    Debugging is the process of testing, finding, and reducing bugs (errors) in computer programs.  
The first known computer bug was a real bug (an insect), stuck in the electronics.**

**J4**

**JavaScript is a programming language. The Syntax rules define how the language is constructed.  
  
JavaScript Syntax  
JavaScript is a scripting language. It is a lightweight, but powerful, programming language.  
  
Syntax definition: "The principles by which sentences are constructed in a language."  
  
The sentences of a programming language are called computer statements, or just statements.  
  
JavaScript Literals  
In a programming language, a literal is a constant value, like 3.14.  
  
Number literals can be written with or without decimals, and with or without scientific notation (e):  
  
3.14  
  
1001  
  
123e5  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML = 123e5;  
< /script>  
  
< /body>  
< /html>  
  
String literals can be written with double or single quotes:  
  
"John Doe"  
  
'John Doe'  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML = 'John Doe';  
< /script>  
  
< /body>  
< /html>  
  
Expression literals evaluates (computes) to a value:  
  
5 + 6  
  
5 \* 10  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML = 5 \* 10;  
< /script>  
  
< /body>  
< /html>  
  
Array literals defines an array:  
  
[40, 100, 1, 5, 25, 10]  
Object literals defines an object:  
  
{firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}  
Function literals defines a function:  
  
function myFunction(a, b) { return a \* b;}  
  
JavaScript Variables  
In a programming language (and in normal algebra), named variables store data values.  
  
JavaScript uses the var keyword to define variables, and an equal sign to assign values to variables (just like algebra):  
  
var x, length  
  
x = 5  
  
length = 6  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var length;  
length = 6;  
document.getElementById("demo").innerHTML = length;  
< /script>  
  
< /body>  
< /html>  
  
A variable can have variable values during the execution of a JavaScript. A literal is always a constant value.  
  
Note    A variable is a name. A literal is value.  
JavaScript Operators  
JavaScript uses arithmetic operators to compute values (just like algebra):  
  
(5 + 6) \* 10  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML = (5 + 6) \* 10;  
< /script>  
  
< /body>  
< /html>  
  
JavaScript uses an assignment operator to assign values to variables (just like algebra):  
  
x = 5  
y = 6  
z = (x + y) \* 10  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x, y, z;  
x = 5  
y = 6;  
z = (x + y) \* 10;  
document.getElementById("demo").innerHTML = z;  
< /script>  
  
< /body>  
< /html>  
  
The JavaScript language has many types of operators:  
  
Type    Examples    Description  
Assignment, arithmetic, and bitwise operators    =  +  -  \*  /    Described in JS Operators  
Conditional, comparison, and logical operators    ==  != <  >     Described in JS Comparisons  
  
JavaScript Statements  
In HTML, JavaScript statements are written as sequences of "commands" to the HTML browser.  
  
Statements are separated by semicolons:  
  
x = 5 + 6;  
y = x \* 10;  
  
JavaScript Keywords  
A JavaScript statement often starts with a keyword. The var keyword tells the browser to create a new variable:  
  
var x = 5 + 6;  
var y = x \* 10;  
  
JavaScript Identifiers  
All programming languages must identify variables, functions, and objects, with unique names.  
  
These unique names are called identifiers.  
  
Identifier names can contain letters, digits, underscores, and dollar signs, but cannot begin with a number.  
  
Reserved words (like JavaScript keywords) cannot be used as identifiers.  
  
JavaScript Comments  
Not all JavaScript statements are "commands". Anything after double slashes // is ignored by the browser:  
  
// I will not be executed  
  
JavaScript Data Types  
JavaScript variables can hold many types of data: numbers, text strings, arrays, objects and much more:  
  
var length = 16;                               // Number assigned by a number literal  
var points = x \* 10;                           // Number assigned by an expression literal  
var lastName = "Johnson";                      // String assigned by a string literal  
var cars = ["Saab", "Volvo", "BMW"];           // Array assigned by an array literal  
var person = {firstName:John, lastName:Doe};   // Object assigned by an object literal  
Note    We use blue color to highlight reserved words, brown for string literals, and green for comments.  
JavaScript Functions  
JavaScript statements written inside a function, can be invoked many times (reused):  
  
Invoke a function = Call upon a function (ask for the code in the function to be executed).  
  
function myFunction(a, b) {  
       return a \* b;                              // returns the product of a and b  
}  
  
JavaScript is Case Sensitive  
In JavaScript all identifiers are case sensitive.  
  
The variables lastName and lastname, are two different variables.  
  
The functions myFunction and myfunction, are two different functions.  
  
JavaScript does not interpret Var; as var.  
  
JavaScript Character Set  
JavaScript uses the Unicode character set.  
  
Unicode covers (almost) all the characters, punctuations, and symbols in the world.  
  
For a closer look, please study our Complete Unicode Reference.  
  
Did You Know?  
Note  
It is common, in JavaScript, to use camelCase names.  
You will often see identifier names written like lastName (instead of lastname).**

**J5**

**In HTML, JavaScript statements are command lines executed by the web browser.  
  
JavaScript Statements  
In HTML, JavaScript statements are "commands" to the browser.  
  
The purpose, of the statements, is to tell the browser what to do.  
  
This JavaScript statement tells the browser to write "Hello Dolly" inside an HTML element identified with id="demo":  
  
Example  
  
document.getElementById("demo").innerHTML = "Hello Dolly.";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My Web Page</h1>  
  
< p id="demo">My first paragraph.</p>  
  
< script>  
document.getElementById("demo").innerHTML = "Hello Dolly.";  
< /script>  
  
< /body>  
< /html>  
  
  
Semicolon ;  
Semicolon separates JavaScript statements.  
  
Normally you add a semicolon at the end of each executable statement.  
  
Using semicolons also makes it possible to write many statements on one line.  
  
Writing:  
  
a = 5;  
b = 6;  
c = a + b;  
Is the same as writing:  
  
a = 5; b = 6; c = a + b;  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My Web Page</h1>  
  
< p id="demo1"></p>  
< p id="demo2"></p>  
  
< script>  
a = 1;  
b = 2;  
c = a + b;  
document.getElementById("demo1").innerHTML = c;  
x = 1; y = 2; z = x + y;  
document.getElementById("demo2").innerHTML = z;  
< /script>  
  
< /body>  
< /html>  
  
  
Note    You might see examples without semicolons.  
Ending statements with semicolon is optional in JavaScript.  
  
JavaScript Code  
JavaScript code (or just JavaScript) is a sequence of JavaScript statements.  
  
Each statement is executed by the browser in the sequence they are written.  
  
This example will manipulate two different HTML elements:  
  
Example  
  
document.getElementById("demo").innerHTML = "Hello Dolly.";  
document.getElementById("myDiv").innerHTML = "How are you?";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My Web Page</h1>  
  
< p id="demo">A Paragraph.</p>  
  
< div id="myDiv">An HTML div.</div>  
  
< script>  
document.getElementById("demo").innerHTML = "Hello Dolly.";  
document.getElementById("myDiv").innerHTML = "How are you?";  
< /script>  
  
< /body>  
< /html>  
  
  
JavaScript Code Blocks  
JavaScript statements can be grouped together in blocks.  
  
Blocks start with a left curly bracket, and end with a right curly bracket.  
  
The purpose of a block is to make the sequence of statements execute together.  
  
A good example of statements grouped together in blocks, are in JavaScript functions.  
  
This example will run a function that will manipulate two HTML elements:  
  
Example  
  
function myFunction() {  
    document.getElementById("demo").innerHTML = "Hello Dolly.";  
    document.getElementById("myDIV").innerHTML = "How are you?";  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My Web Page</h1>  
  
< p id="myPar">I am a paragraph.</p>  
< div id="myDiv">I am a div.</div>  
  
< p>  
< button type="button" onclick="myFunction()">Try it</button>  
< /p>  
  
< script>  
function myFunction() {  
    document.getElementById("myPar").innerHTML = "Hello Dolly.";  
    document.getElementById("myDiv").innerHTML = "How are you?";  
}  
< /script>  
  
< p>When you click on "Try it", the two elements will change.</p>  
  
< /body>  
< /html>  
  
Note    In this tutorial we use 4 spaces of indentation for code blocks.  
You will learn much more about functions later in this tutorial.  
  
JavaScript Statement Identifiers  
JavaScript statements often start with a statement identifier to identify the JavaScript action to be performed.  
  
Statement identifiers are reserved words and cannot be used as variable names (or any other things).  
  
Here is a list of some of the JavaScript statements (reserved words) you will learn about in this tutorial:  
  
Statement    Description  
break    Terminates a switch or a loop.  
catch    Marks the block of statements to be executed when an error occurs in a try block.  
continue    Jumps out of a loop and starts at the top.  
do ... while    Executes a block of statements and repeats the block while a condition is true.  
for    Marks a block of statements to be executed as long as a condition is true.  
for ... in    Marks a block of statements to be executed for each element of an object (or array).  
function    Declares a function.  
if ... else    Marks a block of statements to be executed depending on a condition.  
return    Exits a function.  
switch    Marks a block of statements to be executed depending on different cases.  
throw    Throws (generates) an error.  
try    Implements error handling to a block of statements.  
var    Declares a variable.  
while    Marks a block of statements to be executed while a condition is true.  
There is a complete list of reserved words, in a later chapter of this tutorial.  
  
JavaScript White Space  
JavaScript ignores extra spaces. You can add white space to your script to make it more readable.  
  
The following lines are equivalent:  
  
var person = "Hege";  
var person="Hege";  
  
JavaScript Line Length and Line Breaks  
For best readability, programmers often like to avoid lines longer than 80 characters.  
  
If a JavaScript statement does not fit on one line, the best place to break it, is after an operator or a comma.  
  
Example  
  
document.getElementById("demo").innerHTML =  
    "Hello Dolly.";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My Web Page</h1>  
  
< p>  
The best place to break a code line is after an operator or a comma.  
< /p>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML =  
    "Hello Dolly.";  
< /script>  
  
< /body>  
< /html>  
  
  
JavaScript Line Break in a String  
You can break up a code line within a text string with a backslash:  
  
Example  
  
document.getElementById("demo").innerHTML = "Hello \  
    Dolly!";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My Web Page</h1>  
  
< p>  
You can break a code line within a text string with a backslash.  
< /p>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML = "Hello \  
    Dolly.";  
< /script>  
  
< /body>  
< /html>  
  
However, you cannot break up a code line like this:  
  
Example  
  
document.getElementById("demo").innerHTML = \  
    "Hello Dolly!";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My Web Page</h1>  
  
< p id="demo">You cannot break a code line with a \ (backslash).</p>  
  
< script>  
document.getElementById("demo").innerHTML = \  
    "Hello Dolly.";  
< /script>  
  
< /body>  
< /html>**

**J6**

**JavaScript comments can be used to explain the code, and make the code more readable.  
  
JavaScript comments can also be used to prevent execution, when testing alternative code.  
  
Single Line Comments  
Single line comments start with //.  
  
Any text between // and the end of a line, will be ignored by JavaScript (will not be executed).  
  
The following example uses a single line comment in front of each line, to explain the code:  
  
Example  
  
// Change heading:  
document.getElementById("myH").innerHTML = "My First Page";  
// Change paragraph:  
document.getElementById("myP").innerHTML = "My first paragraph.";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1 id="myH"></h1>  
< p id="myP"></p>  
  
< script>  
// Change heading:  
document.getElementById("myH").innerHTML = "My First Page";  
// Change paragraph:  
document.getElementById("myP").innerHTML = "My first paragraph.";  
< /script>  
  
< p><strong>Note:</strong> The comments are not executed.</p>  
  
< /body>  
< /html>  
  
This example uses a single line comment at the end of each line, to explain the code:  
  
Example  
  
var x = 5;      // Declare x, give it the value of 5  
var y = x + 2;  // Declare y, give it the value of x + 2  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x = 5;      // Declare x, give it the value of 5  
var y = x + 2;  // Declare y, give it the value of x + 2  
  
document.getElementById("demo").innerHTML = y; // Write y to demo  
< /script>  
  
< p><strong>Note:</strong> The comments are not executed.</p>  
  
< /body>  
< /html>  
  
  
Multi-line Comments  
Multi-line comments start with /\* and end with \*/.  
  
Any text between /\* and \*/ will be ignored by JavaScript.  
  
The following example uses a multi-line comment (a comment block) to explain the code:  
  
Example  
  
/\*  
The code below will change  
the heading with id = "myH"  
and the paragraph with id = "myp"  
in my web page:  
\*/  
document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1 id="myH"></h1>  
< p id="myP"></p>  
  
< script>  
/\*  
The code below will change  
the heading with id = "myH"  
and the paragraph with id = "myp"  
in my web page:  
\*/  
document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";  
< /script>  
  
< p><strong>Note:</strong> The comment block is not executed.</p>  
  
< /body>  
< /html>  
  
Note    It is most common to use single line comments.  
Block comments are often used for formal documentation.  
Using Comments to Prevent Execution  
Using comments to prevent execution of code, can be very suitable for testing.  
  
Adding // in front of a code line changes the code lines from an executable line to a comment.  
  
The next example uses // to prevent execution of one of the code lines.  
  
Example  
  
//document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1 id="myH"></h1>  
  
< p id="myP"></p>  
  
< script>  
//document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";  
< /script>  
  
< p><strong>Note:</strong> The comment is not executed.</p>  
  
< /body>  
< /html>  
  
The following example uses a comment block to prevent execution of multiple lines:  
  
Example  
  
/\*  
document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";  
\*/  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1 id="myH"></h1>  
  
< p id="myP"></p>  
  
< script>  
/\*  
document.getElementById("myH").innerHTML = "Welcome to my Homepage";  
document.getElementById("myP").innerHTML = "This is my first paragraph.";  
\*/  
< /script>  
  
< p><strong>Note:</strong> The comment-block is not executed.</p>  
  
< /body>  
< /html>**

**J7**

**JavaScript variables are "containers" for storing information:  
  
Example  
  
var x = 5;  
var y = 6;  
var z = x + y;  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x = 5;  
var y = 6;  
document.getElementById("demo").innerHTML = x + y;  
//document.getElementById("demo").innerHTML = x;  
//document.getElementById("demo").innerHTML = y;  
< /script>  
  
< p>Try to experiment with the // comments.</p>  
  
< /body>  
< /html>  
  
  
Much Like Algebra  
x = 5  
y = 6  
z = x + y  
  
In algebra we use letters (like x) to hold values (like 5).  
  
From the expression z = x + y above, we can calculate the value of z to be 11.  
  
In JavaScript these letters are called variables.  
  
Note    JavaScript variables are containers for storing data.  
  
JavaScript Variables  
As with algebra, JavaScript variables can be used to hold values (x = 5) or expressions (z = x + y).  
  
Variable can have short names (like x and y) or more descriptive names (age, sum, totalVolume).  
  
Variable names can contain letters, digits, underscores, and dollar signs.  
  
Variable names must begin with a letter  
Variable names can also begin with $ and \_ (but we will not use it)  
Variable names are case sensitive (y and Y are different variables)  
Reserved words (like JavaScript keywords) cannot be used as variable names  
Note    Both JavaScript statements and JavaScript variables are case-sensitive.  
  
The Assignment Operator  
In JavaScript, the equal sign (=) is an "assignment" operator, is not an "equal to" operator.  
  
This is different from algebra. The following does not make any sense in algebra:  
  
x = x + 5  
In JavaScript, however it makes perfect sense: Assign the value of x + 5 to the variable x.  
  
In reality: Calculate the value of x + 5. Then put the result into the variable x.  
  
Note    The "equal to" operator in JavaScript, is written like == or ===. You will see it soon!.  
  
JavaScript Data Types  
JavaScript variables can hold many types of data, like text values (person = "John Doe").  
  
In JavaScript texts are called strings or text strings.  
  
There are many types of JavaScript variables, but for now, just think of numbers and strings.  
  
When you assign a string value to a variable, you put double or single quotes around the value.  
  
When you assign a numeric value to a variable, you do not put quotes around the value.  
  
If you put quotes around a numeric value, it will be treated as a text string.  
  
Example  
  
var pi = 3.14;  
var person = "John Doe";  
var answer = 'Yes I am!';  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Create variables of different types:</p>  
  
< p id="demo"></p>  
  
< script>  
var pi = 3.14;  
var person = "John Doe";  
var answer = 'Yes I am!';  
  
//document.getElementById("demo").innerHTML = pi;  
document.getElementById("demo").innerHTML = person;  
//document.getElementById("demo").innerHTML = answer;  
< /script>  
  
< p>Try to experiment with the // comments.</p>  
  
< /body>  
< /html>  
  
  
Declaring (Creating) JavaScript Variables  
Creating a variable in JavaScript is called "declaring" a variable.  
  
You declare JavaScript variables with the var keyword:  
  
var carName;  
After the declaration, the variable is empty (it has no value).  
  
To assign a value to the variable, use the equal sign:  
  
carName = "Volvo";  
You can also assign a value to the variable when you declare it:  
  
var carName = "Volvo";  
In the example below, we create a variable called carName and assign the value "Volvo" to it.  
  
Then we "output" the value inside an HTML paragraph with id="demo":  
  
Example  
  
< p id="demo"></p>  
  
< script>  
var carName = "Volvo";  
document.getElementById("demo").innerHTML = carName;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Create a variable, assign a value to it, and display it:</p>  
  
< p id="demo"></p>  
  
< script>  
var carName = "Volvo";  
document.getElementById("demo").innerHTML = carName;  
< /script>  
  
< /body>  
< /html>  
  
  
Note    It's a good programming practice to declare all variables at the beginning of a script.  
  
One Statement, Many Variables  
You can declare many variables in one statement.  
  
Start the statement with var and separate the variables by comma:  
  
var lastName = "Doe", age = 30, job = "carpenter";  
Your declaration can also span multiple lines:  
  
var lastName = "Doe",  
age = 30,  
job = "carpenter";  
In JavaScript you can always separate statements by semicolon, but then you cannot omit the var keyword.  
  
Wrong:  
  
var lastName = "Doe"; age = 30; job = "carpenter";  
Right;  
  
var lastName = "Doe"; var age = 30; var job = "carpenter";  
  
Value = undefined  
In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input. Variable declared without a value will have the value undefined.  
  
The variable carName will have the value undefined after the execution of the following statement:  
  
var carName;  
  
Re-Declaring JavaScript Variables  
If you re-declare a JavaScript variable, it will not lose its value:.  
  
The value of the variable carName will still have the value "Volvo" after the execution of the following two statements:  
  
var carName = "Volvo";  
var carName;  
  
JavaScript Arithmetic  
As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:  
  
Example  
  
var y = 5;  
var x = y + 2;  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Assign 5 to y, and display the result of y + 2:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var y = 5;  
    document.getElementById("demo").innerHTML = y + 2;  
}  
< /script>  
  
< /body>  
< /html>  
  
You can also add strings, but strings will be concatenated (added end-to-end):  
  
Example  
  
var y = "5";  
var x = y + 2;  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Assign 5 to y, and display the result of y + 2:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var y = "5";  
    document.getElementById("demo").innerHTML = y + 2;  
}  
< /script>  
  
< /body>  
< /html>  
  
Note that if you add a number to a string, both will be treated as strings.  
  
You will learn a lot more about arithmetic operators later in this tutorial.**J8

**String, Number, Boolean, Array, Object, Null, Undefined.  
JavaScript Has Dynamic Types  
JavaScript has dynamic types. This means that the same variable can be used as different types:  
  
Example  
  
var x;               // Now x is undefined  
var x = 5;           // Now x is a Number  
var x = "John";      // Now x is a String  
  
JavaScript Strings  
A string is a variable which stores a series of characters like "John Doe".  
  
Strings are written with quotes. You can use single or double quotes:  
Example  
  
var carName = "Volvo XC60";   // Using double quotes  
var carName = 'Volvo XC60';   // Using single quotes  
You can use quotes inside a string, as long as they don't match the quotes surrounding the string:  
  
Example  
  
var answer = "It's alright";             // Single quote inside double quotes  
var answer = "He is called 'Johnny'";    // Single quotes inside double quotes  
var answer = 'He is called "Johnny"';    // Double quotes inside single quotes  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var carName1 = "Volvo XC60";  
var carName2 = 'Volvo XC60';  
var answer1 = "It's alright";  
var answer2 = "He is called 'Johnny'";  
var answer3 = 'He is called "Johnny"';  
  
document.getElementById("demo").innerHTML =  
carName1 + "<br>" +  
carName2 + "<br>" +  
answer1 + "<br>" +  
answer2 + "<br>" +  
answer3;  
< /script>  
  
< /body>  
< /html>  
  
  
You will learn a lot more about strings later in this tutorial.  
  
JavaScript Numbers  
JavaScript has only one type of numbers.  
  
Numbers can be written with, or without decimals:  
Example  
  
var x1 = 34.00;      // Written with decimals  
var x2 = 34;         // Written without decimals  
Extra large or extra small numbers can be written with scientific (exponential) notation:  
  
Example  
  
var y = 123e5;       // 12300000  
var z = 123e-5;      // 0.00123  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x1 = 34.00;  
var x2 = 34;  
var y = 123e5;  
var z = 123e-5;  
  
document.getElementById("demo").innerHTML = x1 + "<br>" + x2 + "<br>" + y + "<br>" + z  
< /script>  
  
< /body>  
< /html>  
  
  
You will learn a lot more about numbers in the advanced section of this tutorial.  
  
JavaScript Booleans  
Booleans can only have two values: true or false.  
  
var x = true;  
var y = false;  
Booleans are often used in conditional testing.  
  
You will learn a lot more about conditional testing later in this tutorial.  
  
JavaScript Arrays  
JavaScript arrays are written with square brackets.  
  
Array items are separated by commas.  
  
The following code declares (creates) an array called cars, containing three items (car names):  
  
Example  
  
var cars = ["Saab", "Volvo", "BMW"];  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var cars = ["Saab","Volvo","BMW"];  
  
document.getElementById("demo").innerHTML = cars[0];  
< /script>  
  
< /body>  
< /html>  
  
  
Array indexes are zero-based, which means the first item is [0], second is [1], and so on.  
  
You will learn a lot more about arrays later in this tutorial.  
  
JavaScript Objects  
JavaScript objects are written with curly braces.  
  
Object properties are written as name:value pairs, separated by commas.  
  
Example  
  
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
  
< !DOCTYPE html>  
< html>  
< body>  
< p id="demo"></p>  
  
< script>  
var person = {  
    firstName : "John",  
    lastName  : "Doe",  
    age       : 50,  
    eyeColor  : "blue"  
};  
  
document.getElementById("demo").innerHTML =  
person.firstName + " is " + person.age + " years old.";  
< /script>  
  
< /body>  
< /html>  
  
The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.  
  
You will learn a lot more about objects later in this tutorial.  
  
Undefined and Null  
The value of a variable with no value is undefined.  
  
Variables can be emptied by setting the value to null.  
  
Example  
  
var cars;              // Value is undefined  
var person = null;     // Value is null  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The value of a variable with no value is <b>undefined</b>.</p>  
< p>Variables can be emptied by setting the value to <b>null</b>.</p>  
  
< p id="demo"></p>  
  
< script>  
var person;  
var car = "Volvo";  
var x = null;  
document.getElementById("demo").innerHTML =  
person + "<br>" + car + "<br>" + x;  
< /script>  
  
< /body>  
< /html>  
  
  
The typeof Operator  
You can use the JavaScript typeof operator to find the type of a JavaScript variable.  
  
Example  
  
typeof "John"                 // Returns string  
typeof 3.14                   // Returns number  
typeof false                  // Returns boolean  
typeof [1,2,3,4]              // Returns object  
typeof {name:'John', age:34}  // Returns object  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The typeof operator returns the type of a variable or an expression.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML =  
    typeof "john" + "<br>" +  
    typeof 3.14 + "<br>" +  
    typeof false + "<br>" +  
    typeof [1,2,3,4] + "<br>" +  
    typeof {name:'john', age:34};  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
Note     In JavaScript, an array is a special type of object. Therefore typeof [1,2,3,4] returns object.  
Do Not Declare String, Number, and Boolean as Objects!  
When a JavaScript variable is declared with the keyword "new", the variable is created as an object:  
  
var x = new String();          // Declares x as a String object  
var y = new Number();          // Declares y as a Number object  
var z = new Boolean();         //    Declares z as a Boolean object  
Note    Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.**

**J9**

**Objects are just data, with added properties and methods.  
  
Object Properties and Methods  
Properties are values associated with objects.  
  
Methods are actions objects can perform.  
  
A Real Life Example. A Car Object.  
In real life, a car is an object. It has properties like weight and color, and methods like start and stop:  
  
Object    Properties    Methods**[**car.name**](http://car.name/) **= Fiat  
  
car.model = 500  
  
car.weight = 850kg  
  
car.color = white  
car.start()  
  
car.drive()  
  
car.brake()  
  
car.stop()  
All cars have the same properties, but the property values differ from car to car.  
  
All cars have the same methods, but they are performed at different times.  
  
  
JavaScript Objects  
In JavaScript, objects are data (variables), with properties and methods.  
  
Almost "everything" in JavaScript are treated as objects. Dates, Arrays, Strings, Functions....  
  
In JavaScript you can also create your own objects.  
  
This example creates an object called "person", and adds four properties to it:  
  
Example  
  
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
  
< !DOCTYPE html>  
< html>  
< body>  
< p id="demo"></p>  
  
< script>  
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
  
document.getElementById("demo").innerHTML =  
person.firstName + " is " + person.age + " years old.";  
< /script>  
  
< /body>  
< /html>  
  
Spaces and line breaks are not important. An object declaration can span multiple lines:  
  
Example  
  
var person = {  
    firstName:"John",  
    lastName:"Doe",  
    age:50,  
    eyeColor:"blue"  
};  
  
< !DOCTYPE html>  
< html>  
< body>  
< p id="demo"></p>  
  
< script>  
var person = {  
    firstName : "John",  
    lastName  : "Doe",  
    age       : 50,  
    eyeColor  : "blue"  
};  
  
document.getElementById("demo").innerHTML =  
person.firstName + " is " + person.age + " years old.";  
< /script>  
  
< /body>  
< /html>  
  
There are many different ways to create new JavaScript objects.  
  
You can also add new properties and methods to already existing objects.  
  
You will learn much more about objects later in this tutorial.  
  
Accessing Object Properties  
You can access the object properties in two ways:  
  
Example  
  
person.lastName;  
person["lastName"];  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var person = {  
    firstName: "John",  
    lastName : "Doe",  
    id       :  5566  
};  
  
document.getElementById("demo").innerHTML = person.firstName + " " + person["lastName"];  
< /script>  
  
< /body>  
< /html>  
  
  
  
Accessing Object Methods  
You can call an object method with the following syntax:  
  
objectName.methodName()  
This example uses the fullName() method of a person object, to get the full name:  
  
Example  
  
name = person.fullName();  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Creating and using an object method:</p>  
  
< p id="demo"></p>  
  
< script>  
var person = {  
    firstName: "John",  
    lastName : "Doe",  
    id       : 5566,  
    fullName : function (){return this.firstName + " " + this.lastName}  
};  
  
document.getElementById("demo").innerHTML = person.fullName();  
< /script>  
< /body>  
< /html>  
  
Object methods are ordinary JavaScript functions defined as object properties.  
  
You will learn much more about functions later in this tutorial.  
  
Note  
Objects written as name value pairs (lastName: "Doe") is similar to:  
  
Associative arrays in PHP.  
Hash tables, hash maps or hashes in C, C++, C#, Java, Perl, and Ruby.  
Dictionaries in Python.**

**J10**

**A JavaScript function is a block of code designed to perform a particular task.  
  
A JavaScript function is executed when "something" invokes it (calls it).  
  
Example  
  
function myFunction(p1, p2) {  
    return p1 \* p2;              // the function returns the product of p1 and p2  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>This example calls a function which performs a calculation, and returns the result:</p>  
  
< p id="demo"></p>  
  
< script>  
function myFunction(a, b) {  
    return a \* b;  
}  
document.getElementById("demo").innerHTML = myFunction(4, 3);  
< /script>  
  
< /body>  
< /html>  
  
  
JavaScript Function Syntax  
A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().  
  
Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).  
  
The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)  
  
The code to be executed, by the function, is placed inside curly brackets: {}  
  
functionName(parameter1, parameter2, parameter3) {  
    code to be executed  
}  
Function parameters are the names listed in the function definition.  
  
Function arguments are the real values received by the function when it is invoked.  
  
Inside the function, the arguments are used as local variables.  
  
Note    A Function is much the same as a Procedure or a Subroutine, in other programming languages.  
  
Function Invocation  
The code inside the function will execute when "something" invokes (calls) the function:  
  
When an event occurs (when a user clicks a button)  
When it is invoked (called) from JavaScript code  
Automatically (self invoked)  
You will learn a lot more about function invocation later in this tutorial.  
  
Function Return  
When JavaScript reaches a return statement, the function will stop executing.  
  
If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.  
  
Functions often compute a return value. The return value is "returned" back to the "caller":  
  
Example  
  
Calculate the product of two numbers, and return the result:  
  
var x = myFunction(4, 3);        // Function is called, return value will end up in x  
  
function myFunction(a, b) {  
    return a \* b;                // Function returns the product of a and b  
}  
The result in x will be:  
  
12  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>This example calls a function which performs a calculation, and returns the result:</p>  
  
< p id="demo"></p>  
  
< script>  
function myFunction(a, b) {  
    return a \* b;  
}  
document.getElementById("demo").innerHTML = myFunction(4, 3);  
< /script>  
  
< /body>  
< /html>  
  
  
Why Functions?  
You can reuse code: Define the code once, and use it many times.  
  
You can use the same code many times with different arguments, to produce different results.  
  
Example  
  
Convert Fahrenheit to Celsius:  
  
function toCelsius(fahrenheit) {  
    return (5/9) \* (fahrenheit-32);  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>This example calls a function to convert from Fahrenheit to Celcius:</p>  
< p id="demo"></p>  
  
< script>  
function toCelcius(f) {  
    return (5/9) \* (f-32);  
}  
document.getElementById("demo").innerHTML = toCelcius(32);  
< /script>  
  
< /body>  
< /html>  
  
  
JavaScript Functions are Objects  
In JavaScript, functions are objects.  
  
JavaScript functions have properties and methods.  
  
You can add your own properties and methods to functions.  
  
JavaScript Functions are Variables Too  
In JavaScript, functions can be used as variables:  
  
Example  
  
Instead of:  
  
temp = toCelsius(32);  
text = "The temperature is " + temp + " Centigrade";  
You can use:  
  
text = "The temperature is " + toCelsius(32) + " Centigrade";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML =  
"The temperature is " + toCelsius(32) + " Centigrade";  
  
function toCelsius(fahrenheit) {  
  return (5/9) \* (fahrenheit-32);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
JavaScript functions can be redefined like ordinary variables.  
  
JavaScript functions can also be passed as values to other functions.  
  
Note    You will learn a lot more about functions later in this tutorial.**

**J11**

**Scope is the set of variables you have access to.  
  
JavaScript Scope  
In JavaScript, objects and functions, are also variables.  
  
In JavaScript, scope is the set of variables, objects, and functions you have access to.  
  
JavaScript has function scope: The scope changes inside functions.  
  
Local JavaScript Variables  
Variables declared within a JavaScript function, become LOCAL to the function.  
  
Local variables have local scope: They can only be accessed within the function.  
  
Example  
  
// code here can not use carName  
  
function myFunction() {  
    var carName = "Volvo";  
  
    // code here can use carName  
  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>A local variable can only be accessed from within the function where it was declared.</p>  
  
< p id="demo"></p>  
  
< script>  
myFunction();  
document.getElementById("demo").innerHTML =  
"I can display " + typeof carName;  
  
function myFunction() {  
    var carName = "Volvo";  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.  
  
Local variables are created when a function starts, and deleted when the function is completed.  
  
Global JavaScript Variables  
A variable declared outside a function, becomes GLOBAL.  
  
A global variable has global scope: All scripts and functions on a web page can access it.  
  
Example  
  
var carName = " Volvo";  
  
// code here can use carName  
  
function myFunction() {  
  
    // code here can use    carName  
  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>A GLOBAL variable can be accessed from any script or function.</p>  
  
< p id="demo"></p>  
  
< script>  
var carName = "Volvo";  
myFunction();  
  
function myFunction() {  
    document.getElementById("demo").innerHTML =  
    "I can display " + carName;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
  
Automatically Global  
If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable.  
  
This code example will declare carName as a global variable, even if it is executed inside a function.  
  
Example  
  
// code here can use carName  
  
function myFunction() {  
    carName = "Volvo";  
  
    // code here can use carName  
  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>  
If you assign a value to a variable that has not been declared,  
it will automatically become a GLOBAL variable:  
< /p>  
  
< p id="demo"></p>  
  
< script>  
myFunction();  
document.getElementById("demo").innerHTML =  
"I can display " + carName;  
  
function myFunction() {  
    carName = "Volvo";  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
The Lifetime of JavaScript Variables  
The lifetime of a JavaScript variable starts when it is declared.  
  
Local variables are deleted when the function is completed.  
  
Global variables are deleted when you close the page.  
  
Function Arguments  
Function arguments (parameters) work as local variables inside functions.  
  
Global Variables in HTML  
With JavaScript, the global scope is the complete JavaScript environment.  
  
In HTML, the global scope is the window object: All global variables belong to the window object.  
  
Example  
  
// code here can use window.carName  
  
function myFunction() {  
    carName = "Volvo";  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>  
In HTML, all global variables will become a window variables.  
< /p>  
  
< p id="demo"></p>  
  
< script>  
myFunction();  
document.getElementById("demo").innerHTML =  
"I can display " + window.carName;  
  
function myFunction() {  
    carName = "Volvo";  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Did You Know?  
Note    Your global variables, or functions, can overwrite window variables or functions.  
Anyone, inclusive the window object, can overwrite your global variables or functions.**

**J12**

**HTML events are "things" that happen to HTML elements.  
  
When JavaScript is used in HTML pages, JavaScript can "react" on these events.  
  
HTML Events  
An HTML event can be something the browser does, or something a user does.  
  
Here are some examples of HTML events:  
  
An HTML web page has finished loading  
An HTML input field was changed  
An HTML button was clicked  
Often, when events happen, you may want to do something.  
  
JavaScript lets you execute code when events are detected.  
  
HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.  
  
With single quotes:  
  
< some-HTML-element some-event='some JavaScript'>  
With double quotes:  
  
< some-HTML-element some-event="some JavaScript">  
In the following example, an onclick attribute (with code), is added to a button element:  
  
Example  
  
< button onclick='getElementById("demo").innerHTML=Date()'>The time is?</button>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< button onclick="getElementById('demo').innerHTML=Date()">The time is?</button>  
  
< p id="demo"></p>  
  
< /body>  
< /html>  
  
In the example above, the JavaScript code changes the content of the element with id="demo".  
  
In the next example, the code changes the content of it's own element (using this.innerHTML):  
  
Example  
  
< button onclick="this.innerHTML=Date()">The time is?</button>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< button onclick="this.innerHTML=Date()">The time is?</button>  
  
< /body>  
< /html>  
  
  
Note    JavaScript code is often several lines long. It is more common to see event attributes calling functions:  
  
Example  
  
< button onclick="displayDate()">The time is?</button>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to display the date.</p>  
  
< button onclick="displayDate()">The time is?</button>  
  
< script>  
function displayDate() {  
    document.getElementById("demo").innerHTML = Date();  
}  
< /script>  
  
< p id="demo"></p>  
  
< /body>  
< /html>  
  
  
Common HTML Events  
Here is a list of some common HTML events:  
  
Event    Description  
onchange    An HTML element has been changed  
onclick    The user clicks an HTML element  
onmouseover    The user moves the mouse over an HTML element  
onmouseout    The user moves the mouse away from an HTML element  
onkeydown    The user pushes a keyboard key  
onload    The browser has finished loading the page  
The list is much longer: W3Schools JavaScript Reference HTML DOM Events.  
  
What can JavaScript Do?  
Event handlers can be used to handle, and verify, user input, user actions, and browser actions:  
  
Things that should be done every time a page loads  
Things that should be done when the page is closed  
Action that should be performed when a user clicks a button  
Content that should be verified when a user input data  
And more ...  
Many different methods can be used to let JavaScript work with events:  
  
HTML event attributes can execute JavaScript code directly  
HTML event attributes can call JavaScript functions  
You can assign your own event handler functions to HTML elements  
You can prevent events from being sent or being handled  
And more ...  
Note    You will learn a lot more about events and event handlers in the HTML DOM chapters.**

**J13**

**JavaScript strings are used for storing and manipulating text.  
  
JavaScript Strings  
A JavaScript string simply stores a series of characters like "John Doe".  
  
A string can be any text inside quotes. You can use single or double quotes:  
  
Example  
  
var carname = "Volvo XC60";  
var carname = 'Volvo XC60';  
You can use quotes inside a string, as long as they don't match the quotes surrounding the string:  
  
Example  
  
var answer = "It's alright";  
var answer = "He is called 'Johnny'";  
var answer = 'He is called "Johnny"';  
Or you can put quotes inside a string by using the \ escape character:  
  
Example  
  
var answer = 'It\'s alright';  
var answer = "He is called \"Johnny\"";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var carName1 = "Volvo XC60";  
var carName2 = 'Volvo XC60';  
var answer1 = "It's alright";  
var answer2 = "He is called 'Johnny'";  
var answer3 = 'He is called "Johnny"';  
  
document.getElementById("demo").innerHTML =  
carName1 + "<br>" +  
carName2 + "<br>" +  
answer1 + "<br>" +  
answer2 + "<br>" +  
answer3;  
< /script>  
  
< /body>  
< /html>  
  
  
String Length  
The length of a string (a string object) is found in the built in property length:  
  
Example  
  
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
var sln = txt.length;  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var txt="ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
document.getElementById("demo").innerHTML = txt.length;  
< /script>  
  
< /body>  
< /html>  
  
  
Special Characters  
In JavaScript, strings are written as characters inside single or double quotes.  
  
Because of this, JavaScript will misunderstand this string:  
  
 "We are the so-called "Vikings" from the north."  
The string will be chopped to "We are the so-called ".  
  
To solve this problem, you can place a backslash (\) before the double quotes in "Vikings":  
  
 "We are the so-called \"Vikings\" from the north."  
 The backslash is an escape character. Escape characters turns special characters into string characters:  
  
The escape character (\) can be used to insert apostrophes, new lines, quotes, and other special characters into a string.  
  
The table below lists other special characters that can be added to a text string with the backslash sign:  
  
Code    Outputs  
\'    single quote  
\"    double quote  
\\    backslash  
\n    new line  
\r    carriage return  
\t    tab  
\b    backspace  
\f    form feed  
  
Strings Can be Objects  
Normally, JavaScript strings are primitive values, created from literals: var firstName = "John"  
  
But strings can also be defined as objects with the keyword new: var firstName = new String("John")  
  
Example  
  
var x = "John";  
var y = new String("John");  
  
// type of x will return String  
// type of y will return Object  
  
< !DOCTYPE html>  
< html>  
< body>  
< p id="demo"></p>  
  
< script>  
var x = "John";              // x is a string  
var y = new String("John");  // y is an object  
  
document.getElementById("demo").innerHTML =  
typeof x + " " + typeof y;  
< /script>  
  
< /body>  
< /html>  
  
Note    Don't create String objects. They slow down execution speed, and produce nasty side effects:  
Example  
  
var x = "John";  
var y = new String("John");  
  
// (x === y) is now false because x is a string and y is an object.  
  
< !DOCTYPE html>  
< html>  
< body>  
< p id="demo"></p>  
  
< script>  
var x = "John";              // x is a string  
var y = new String("John");  // y is an object  
document.getElementById("demo").innerHTML = x===y;  
< /script>  
  
< /body>  
< /html>  
  
  
String Properties and Methods  
Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).  
  
But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.  
  
String methods are covered in next chapter.  
  
String Properties  
Property    Description  
constructor    Returns the function that created the String object's prototype  
length    Returns the length of a string  
prototype    Allows you to add properties and methods to an object  
  
String Methods  
Method    Description  
charAt()    Returns the character at the specified index (position)  
charCodeAt()    Returns the Unicode of the character at the specified index  
concat()    Joins two or more strings, and returns a copy of the joined strings  
fromCharCode()    Converts Unicode values to characters  
indexOf()    Returns the position of the first found occurrence of a specified value in a string  
lastIndexOf()    Returns the position of the last found occurrence of a specified value in a string  
localeCompare()    Compares two strings in the current locale  
match()    Searches a string for a match against a regular expression, and returns the matches  
replace()    Searches a string for a value and returns a new string with the value replaced  
search()    Searches a string for a value and returns the position of the match  
slice()    Extracts a part of a string and returns a new string  
split()    Splits a string into an array of substrings  
substr()    Extracts a part of a string from a start position through a number of characters  
substring()    Extracts a part of a string between two specified positions  
toLocaleLowerCase()    Converts a string to lowercase letters, according to the host's locale  
toLocaleUpperCase()    Converts a string to uppercase letters, according to the host's locale  
toLowerCase()    Converts a string to lowercase letters  
toString()    Returns the value of a String object  
toUpperCase()    Converts a string to uppercase letters  
trim()    Removes whitespace from both ends of a string  
valueOf()    Returns the primitive value of a String object**

**J14**

**String methods help you to work with strings.  
  
Finding a String in a String  
The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string:  
  
Example  
  
var str = "Please locate where 'locate' occurs!";  
var pos = str.indexOf("locate");  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="p1">Please locate where 'locate' occurs!.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var str = document.getElementById("p1").innerHTML;  
    var pos = str.indexOf("locate");  
    document.getElementById("demo").innerHTML = pos;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The lastIndexOf() method returns the index of the last occurrence of a specified text in a string:  
  
Example  
  
var str = "Please locate where 'locate' occurs!";  
var pos = str.lastIndexOf("locate");  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="p1">Please locate where 'locate' occurs!.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var str = document.getElementById("p1").innerHTML;  
    var pos = str.lastIndexOf("locate");  
    document.getElementById("demo").innerHTML = pos;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Both the indexOf(), and the lastIndexOf() methods return -1 if the text is not found.  
  
Note    JavaScript counts positions from zero.  
0 is the first position in a string, 1 is the second, 2 is the third ...  
Both methods accept a second parameter as the starting position for the search.  
  
Searching for a String in a String  
The search() method searches a string for a specified value and returns the position of the match:  
  
Example  
  
var str = "Please locate where 'locate' occurs!";  
var pos = str.search("locate");  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="p1">Please locate where 'locate' occurs!.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var str = document.getElementById("p1").innerHTML;  
    var pos = str.search("locate");  
    document.getElementById("demo").innerHTML = pos;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Note  
Did You Notice?  
The two methods, indexOf() and search(), are equal.  
  
They accept the same arguments (parameters), and they return the same value.  
  
The two methods are equal, but the search() method can take much more powerful search values.  
  
You will learn more about powerful search values in the chapter about regular expressions.  
  
Extracting String Parts  
There are 3 methods for extracting a part of a string:  
  
slice(start, end)  
substring(start, end)  
substr(start, length)  
The slice() Method  
slice() extracts a part of a string and returns the extracted part in a new string.  
  
The method takes 2 parameters: the starting index (position), and the ending index (position).  
  
This example slices out a portion of a string from position 7 to position 13:  
  
Example  
  
var str = "Apple, Banana, Kiwi";  
var res = str.slice(7,13);  
The result of res will be:  
  
Banana  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The slice() method extract a part of a string  
and returns the extracted parts in a new string:</p>  
  
< p id="demo"></p>  
  
< script>  
var str = "Apple, Banana, Kiwi";  
document.getElementById("demo").innerHTML = str.slice(7,13);  
< /script>  
  
< /body>  
< /html>  
  
  
  
If a parameter is negative, the position is counted from the end of the string.  
  
This example slices out a portion of a string from position -12 to position -6:  
  
Example  
  
var str = "Apple, Banana, Kiwi";  
var res = str.slice(-12,-6);  
The result of res will be:  
  
Banana  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The slice() method extract a part of a string  
and returns the extracted parts in a new string:</p>  
  
< p id="demo"></p>  
  
< script>  
var str = "Apple, Banana, Kiwi";  
document.getElementById("demo").innerHTML = str.slice(-12,-6);  
< /script>  
  
< /body>  
< /html>  
  
  
  
If you omit the second parameter, the method will slice out the rest of the sting:  
  
Example  
  
var res = str.slice(7);  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The slice() method extract a part of a string  
and returns the extracted parts in a new string:</p>  
  
< p id="demo"></p>  
  
< script>  
var str = "Apple, Banana, Kiwi";  
document.getElementById("demo").innerHTML = str.slice(7);  
< /script>  
  
< /body>  
< /html>  
  
  
  
or, counting from the end:  
  
Example  
  
var res = str.slice(-12);  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The slice() method extract a part of a string  
and returns the extracted parts in a new string:</p>  
  
< p id="demo"></p>  
  
< script>  
var str = "Apple, Banana, Kiwi";  
document.getElementById("demo").innerHTML = str.slice(-12);  
< /script>  
  
< /body>  
< /html>  
  
  
  
Note    Negative positions does not work in Internet Explorer 8 and earlier.  
The substring() Method  
substring() is similar to slice().  
  
The difference is that substring() cannot accept negative indexes.  
  
Example  
  
var str = "Apple, Banana, Kiwi";  
var res = str.substring(7,13);  
The result of res will be:  
  
Banana  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The substr() method extract a part of a string  
and returns the extracted parts in a new string:</p>  
  
< p id="demo"></p>  
  
< script>  
var str = "Apple, Banana, Kiwi";  
document.getElementById("demo").innerHTML = str.substring(7,13);  
< /script>  
  
< /body>  
< /html>  
  
  
  
If you omit the second parameter, substring() will slice out the rest of the string.  
  
The substr() Method  
substr() is similar to slice().  
  
The difference is that the second parameter specifies the length of the extracted part.  
  
Example  
  
var str = "Apple, Banana, Kiwi";  
var res = str.substr(7,6);  
The result of res will be:  
  
Banana  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The substr() method extract a part of a string  
and returns the extracted parts in a new string:</p>  
  
< p id="demo"></p>  
  
< script>  
var str = "Apple, Banana, Kiwi";  
document.getElementById("demo").innerHTML = str.substr(7,6);  
< /script>  
  
< /body>  
< /html>  
  
  
  
If the first parameter is negative, the position counts from the end of the string.  
  
The second parameter can not be negative, because it defines the length.  
  
If you omit the second parameter, substr() will slice out the rest of the sting.  
  
Replacing String Content  
The replace() method replaces a specified value with another value in a string:  
  
Example  
  
str = "Please visit Microsoft!";  
var n = str.replace("Microsoft","W3Schools");  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Replace "Microsoft" with "W3Schools" in the paragraph below:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo">Please visit Microsoft!</p>  
  
< script>  
function myFunction() {  
    var str = document.getElementById("demo").innerHTML;  
    var txt = str.replace("Microsoft","W3Schools");  
    document.getElementById("demo").innerHTML = txt;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
The replace() method can also take a regular expression as the search value.  
Converting to Upper and Lower Case  
A string is converted to upper case with toUpperCase():  
  
Example  
  
var text1 = "Hello World!";       // String  
var text2 = text1.toUpperCase();  // text2 is text1 converted to upper  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Convert string to upper case:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo">Hello World!</p>  
  
< script>  
function myFunction() {  
    var text = document.getElementById("demo").innerHTML;  
    document.getElementById("demo").innerHTML = text.toUpperCase();  
}  
< /script>  
  
< /body>  
< /html>  
  
  
A string is converted to lower case with toLowerCase():  
  
Example  
  
var text1 = "Hello World!";       // String  
var text2 = text1.toLowerCase();  // text2 is text1 converted to lower  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Convert string to lower case:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo">Hello World!</p>  
  
< script>  
function myFunction() {  
    var text = document.getElementById("demo").innerHTML;  
    document.getElementById("demo").innerHTML = text.toLowerCase();  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
The concat() Method  
concat() joins two or more strings:  
  
Example  
  
var text1 = "Hello";  
var text2 = "World";  
text3 = text1.concat("    ",text2);  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The concat() method joins two or more strings:</p>  
  
< p id="demo"></p>  
  
< script>  
var text1 = "Hello";  
var text2 = "World!"  
document.getElementById("demo").innerHTML = text1.concat(" ",text2);  
< /script>  
  
< /body>  
< /html>  
  
  
  
The concat() method can be used instead of the plus operator. These two lines do the same:  
  
Example  
  
var text = "Hello" + " " + "World!";  
var text = "Hello".concat(" ","World!");  
Note    All string methods return a new string. They don't modify the original string.  
Formally said: Strings are immutable: Strings cannot be changed, only replaced.  
Extracting String Characters  
There are 2 safe methods for extracting string characters:  
  
charAt(position)  
charCodeAt(position)  
The charAt() Method  
The charAt() method returns the character at a specified index (position) in a string:  
  
Example  
  
var str = "HELLO WORLD";  
str.charAt(0);            // returns H  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The charAt() method returns the character at a given position in a string:</p>  
  
< p id="demo"></p>  
  
< script>  
var str = "HELLO WORLD";  
document.getElementById("demo").innerHTML = str.charAt(0);  
< /script>  
< /body>  
< /html>  
  
  
  
The charCodeAt() Method  
The charCodeAt() method returns the unicode of the character at a specified index in a string:  
  
Example  
  
var str = "HELLO WORLD";  
str.charCodeAt(0);         //    returns 72  
  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The charCodeAt() method returns the unicode of the character at a given position in a string:</p>  
  
< p id="demo"></p>  
  
< script>  
var str = "HELLO WORLD";  
document.getElementById("demo").innerHTML = str.charCodeAt(0);  
< /script>  
< /body>  
< /html>  
  
  
Accessing a String as an Array is Unsafe  
You might have seen code like this, accessing a string as an array:  
  
var str = "HELLO WORLD";  
  
str[0];                   // returns H  
This is unsafe and unpredictable:  
  
It does not work in all browsers (not in IE5, IE6, IE7)  
It makes strings look like arrays (but they are not)  
str[0] = "H" does not give an error (but does not work)  
If you want to read a string as an array, convert it to an array first.  
  
Converting a String to an Array  
A string can be converted to an array with the split() method:  
  
Example  
  
var txt = "a,b,c,d,e";   // String  
txt.split(",");          // Split on commas  
txt.split(" ");          // Split on spaces  
txt.split("|");          // Split on pipe  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click "Try it" to display the first array element, after a string split.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var str = "a,b,c,d,e,f";  
    var arr = str.split(",");  
    document.getElementById("demo").innerHTML = arr[0];  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
If the separator is omitted, the returned array will contain the whole string in index [0].  
  
If the separator is "", the returned array will be an array of single characters:  
  
Example  
  
var txt = "Hello";       // String  
txt.split("");           // Split in characters  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var str = "Hello";  
var arr = str.split("");  
var text = "";  
var i;  
for (i = 0; i < arr.length; i++) {  
    text += arr[i] + "<br>"  
}  
document.getElementById("demo").innerHTML = text;  
< /script>  
  
< /body>  
< /html>  
  
  
  
Complete String Reference  
For a complete reference, go to our Complete JavaScript String Reference.  
  
The reference contains descriptions and examples of all string properties and methods.**

**J15**

**JavaScript has only one type of number.  
  
Numbers can be written with, or without, decimals.  
  
JavaScript Numbers  
JavaScript numbers can be written with, or without decimals:  
Example  
  
var x = 3.14;     // A number with decimals  
var y = 34;       // A number without decimals  
Extra large or extra small numbers can be written with scientific (exponent) notation:  
  
Example  
  
var x = 123e5;    // 12300000  
var y = 123e-5;   // 0.00123  
  
JavaScript Numbers are Always 64-bit Floating Point  
Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.  
  
JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.  
  
This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:  
  
Value (aka Fraction/Mantissa)    Exponent    Sign  
52 bits (0 - 51)     11 bits (52 - 62)    1 bit (63)  
  
Precision  
Integers (numbers without a period or exponent notation) are considered accurate up to 15 digits.  
  
Example  
  
var x = 999999999999999;   // x will be 999999999999999  
var y = 9999999999999999;  // y will be 10000000000000000  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Integers are considered accurate up to 15 digits.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var x = 999999999999999;  
    var y = 9999999999999999;  
    document.getElementById("demo").innerHTML = x + "<br>" + y;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The maximum number of decimals is 17, but floating point arithmetic is not always 100% accurate:  
  
Example  
  
var x = 0.2 + 0.1;         // x will be 0.30000000000000004  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Floating point arithmetic is not always 100% accurate.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    x = 0.2 + 0.1;  
    document.getElementById("demo").innerHTML = "0.2 + 0.1 = " + x;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
To solve the problem above, it helps to multiply and divide:  
  
Example  
  
var x = (0.2 \* 10 + 0.1 \* 10) / 10;       // x will be 0.3  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Floating point arithmetic is not always 100% accurate.</p>  
< p>But it helps to multiply and divide.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    x = (0.2\*10 + 0.1\*10) / 10;  
    document.getElementById("demo").innerHTML = "0.2 + 0.1 = " + x;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Hexadecimal  
JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.  
  
Example  
  
var x = 0xFF;             // x will be 255  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Numeric constants, preceded by 0x, are interpreted as hexadecimal.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = "0xFF = " + 0xFF;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Note    Never write a number with a leading zero (like 07).  
Some JavaScript versions interpret numbers as octal if they are written with a leading zero.  
By default, Javascript displays numbers as base 10 decimals.  
  
But you can use the toString() method to output numbers as base 16 (hex), base 8 (octal), or base 2 (binary).  
  
Example  
  
var myNumber = 128;  
myNumber.toString(16);     // returns 80  
myNumber.toString(8);      // returns 200  
myNumber.toString(2);      // returns 10000000  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The toString() method can output numbers as base 16 (hex), base 8 (octal), or base 2 (binary).</p>  
  
< p id="demo"></p>  
  
< button onclick="myFunction()">Try it</button>  
  
< script>  
function myFunction() {  
    var myNumber = 128;  
    document.getElementById("demo").innerHTML = "128 = " +  
    myNumber + " Decimal, " +  
    myNumber.toString(16) + " Hexadecimal, " +  
    myNumber.toString(8) + " Octal, " +  
    myNumber.toString(2) + " Binary."  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Infinity  
Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.  
  
Example  
  
var myNumber = 2;  
while (myNumber != Infinity) {          // Execute until Infinity  
    myNumber = myNumber \* myNumber;  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Infinity is returned if you calculate a number outside the largest possible number.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var myNumber = 2;  
    var txt = "";  
    while (myNumber != Infinity) {  
        myNumber = myNumber \* myNumber;  
        txt = txt + myNumber + "<br>";  
    }  
    document.getElementById("demo").innerHTML = txt;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Division by 0 (zero) also generates Infinity:  
  
Example  
  
var x =  2 / 0;          // x will be Infinity  
var y = -2 / 0;          // y will be -Infinity  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Division by zero also generates Infinity.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var x = 2/0;  
    var y = -2/0;  
    document.getElementById("demo").innerHTML = x + "<br>" + y;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Infinity is a number: typeOf Infinity returns number.  
  
Example  
  
typeof Infinity;        // returns "number"  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Infinity is a Number.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = typeof Infinity;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
NaN - Not a Number  
NaN is a JavaScript reserved word indicating that a value is not a number.  
  
You can use the global JavaScript function isNaN() to find out if a value is a number.  
  
Example  
  
var x = 100 / "Apple";  // a number divided by a string is not a number  
var y = 100 / "10";     // a number divided by a numeric string is a number  
  
< !DOCTYPE html>  
< html>  
< body>  
< p>A number divided by a string is not a number</p>  
< p>A number divided by a numeric string is a number</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var x = 1000 / "Apple";  
    var y = 1000 / "10";  
    document.getElementById("demo").innerHTML = x + "<br>" + y;  
}  
< /script>  
  
< /body>  
< /html>  
  
Infinity is a number.  
  
Example  
  
isNaN(1000 / 0);        // returns false  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Infinity is a number</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
  document.getElementById("demo").innerHTML = isNaN(1000 / 0);  
}  
< /script>  
< p id="demo"></p>  
  
< /body>  
< /html>  
  
Watch out for NaN. If you use it in a mathematical operation, the result will also be NaN.  
  
Numbers Can be Objects  
Normally JavaScript numbers are primitive values created from literals: var x = 123  
  
But numbers can also be defined as objects with the keyword new: var y = new Number(123)  
  
Example  
  
var x = 123;  
var y = new Number(123);  
  
typeof x;               // returns number  
typeof y;               // returns object  
  
< !DOCTYPE html>  
< html>  
< body>  
< p id="demo"></p>  
  
< script>  
var x = 123;  
var y = new Number(123);  
  
document.getElementById("demo").innerHTML = typeof x + "<br>" + typeof y;  
< /script>  
  
< /body>  
< /html>  
  
Note    Don't create Number objects. They slow down execution speed, and produce nasty side effects:  
Example  
  
var x = 123;  
var y = new Number(123);  
(x === y) // is false because x is a number and y is an object.  
  
< !DOCTYPE html>  
< html>  
< body>  
< p id="demo"></p>  
  
< script>  
var x = 123;              // x is a number  
var y = new Number(123);  // y is an object  
document.getElementById("demo").innerHTML = x===y;  
< /script>  
  
< /body>  
< /html>  
  
  
Number Properties and Methods  
Primitive values (like 3.14 or 2014), cannot have properties and methods (because they are not objects).  
  
But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.  
  
Number Properties  
Property    Description  
MAX\_VALUE    Returns the largest number possible in JavaScript  
MIN\_VALUE    Returns the smallest number possible in JavaScript  
NEGATIVE\_INFINITY    Represents negative infinity (returned on overflow)  
NaN    Represents a "Not-a-Number" value  
POSITIVE\_INFINITY    Represents infinity (returned on overflow)  
  
Example  
  
var x = Number.MAX\_VALUE;  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML = Number.MAX\_VALUE;  
< /script>  
  
< /body>  
< /html>  
  
  
Number properties belongs to the JavaScript's number object wrapper called Number.  
  
These properties can only be accessed as Number.MAX\_VALUE.  
  
Using myNumber.MAX\_VALUE, where myNumber is a variable, expression, or value, will return undefined:  
  
Example  
  
var x = 6;  
var y = x.MAX\_VALUE;    // y becomes undefined  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x = 6;  
document.getElementById("demo").innerHTML = x.MAX\_VALUE;  
< /script>  
  
< /body>  
< /html>  
  
  
Note    Number methods are covered in the next chapter**

**J16**

**Numbers methods help you to work with numbers.  
  
Global Methods  
JavaScript global functions can be used on all JavaScript data types.  
  
These are the most relevant methods, when working with numbers:  
  
Method    Description  
Number()    Returns a number, converted from its argument.  
parseFloat()    Parses its argument and returns a floating point number  
parseInt()    Parses its argument and returns an integer  
  
Number Methods  
JavaScript number methods are methods that can be used on numbers:  
  
Method    Description  
toString()    Returns a number as a string  
toExponential()    Returns a string, with a number rounded and written using exponential notation.  
toFixed()    Returns a string, with a number rounded and written with a specified number of decimals.  
toPrecision()    Returns a string, with a number written with a specified length  
valueOf()    Returns a number as a number  
Note    All number methods return a new variable. They do not change the original variable.  
The toString() Method  
toString() returns a number as a string.  
  
All number methods can be used on any type of numbers, literals, variables, or expressions:  
  
Example  
  
var x = 123;  
x.toString();            // returns 123 from variable x  
(123).toString();        // returns 123 from literal 123  
(100 + 23).toString();   // returns 123 from expression 100 + 23  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The toString() method converts a number to a string.</p>  
  
< p id="demo"></p>  
  
< script>  
var x = 123;  
document.getElementById("demo").innerHTML =  
    x.toString() + "<br>" +  
   (123).toString() + "<br>" +  
   (100 + 23).toString();�  
</script>  
< /body>  
< /html>  
  
  
  
The toExponential() Method  
toExponential() returns a string, with a number rounded and written using exponential notation.  
  
A parameter defines the number of character behind the decimal point:  
  
Example  
  
var x = 9.656;  
x.toExponential(2);     // returns 9.66+0  
x.toExponential(4);     // returns 9.6560e+0  
x.toExponential(6);     // returns 9.656000e+0  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The toExponential() method returns a string, with the number rounded and written using exponential notation.</p>  
  
< p>An optional parameter defines the number of digits behind the decimal point.</p>  
  
< p id="demo"></p>  
  
< script>  
var x = 9.656;  
document.getElementById("demo").innerHTML =  
    x.toExponential() + "<br>" +  
    x.toExponential(2) + "<br>" +  
    x.toExponential(4) + "<br>" +  
    x.toExponential(6);  
< /script>  
  
< /body>  
< /html>  
  
  
The parameter is optional. If you don't specify it, JavaScript will not round the number.  
  
The toFixed() Method  
toFixed() returns a string, with the number written with a specified number of decimals:  
  
Example  
  
var x = 9.656;  
x.toFixed(0);           // returns 10  
x.toFixed(2);           // returns 9.66  
x.toFixed(4);           // returns 9.6560  
x.toFixed(6);           // returns 9.656000  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The toFixed() method rounds a number to a given number of digits.</p>  
< p>For working with money, toFixed(2) is perfect.</p>  
  
< p id="demo"></p>  
  
< script>  
var x = 9.656;  
document.getElementById("demo").innerHTML =  
    x.toFixed(0) + "<br>" +  
    x.toFixed(2) + "<br>" +  
    x.toFixed(4) + "<br>" +  
    x.toFixed(6);  
< /script>  
  
< /body>  
< /html>  
  
  
Note    toFixed(2) is perfect for working with money.  
The toPrecision() Method  
toPrecision() returns a string, with a number written with a specified length:  
  
Example  
  
var x = 9.656;  
x.toPrecision();        // returns 9.656  
x.toPrecision(2);       // returns 9.7  
x.toPrecision(4);       // returns 9.656  
x.toPrecision(6);       // returns 9.65600  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The toPrecision() method returns a string, with a number written with a specified length:</p>  
  
< p id="demo"></p>  
  
< script>  
var x = 9.656;  
document.getElementById("demo").innerHTML =  
    x.toPrecision() + "<br>" +  
    x.toPrecision(2) + "<br>" +  
    x.toPrecision(4) + "<br>" +  
    x.toPrecision(6);  
< /script>  
  
< /body>  
< /html>  
  
  
Converting Variables to Numbers  
There are 3 JavaScript functions that can be used to convert variables to numbers:  
  
The Number() method  
The parseInt() method  
The parseFloat() method  
These methods are not number methods, but global JavaScript methods.  
  
The Number() Method  
Number(), can be used to convert JavaScript variables to numbers:  
  
Example  
  
x = true;  
Number(x);        // returns 1  
x = false;  
Number(x);        // returns 0  
x = new Date();  
Number(x);        // returns 1404568027739  
x = "10"  
Number(x);        // returns 10  
x = "10 20"  
Number(x);        // returns NaN  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The global JavaScript function Number() converts variables to numbers:</p>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML =  
    Number(true) + "<br>" +  
    Number(false) + "<br>" +  
    Number(new Date()) + "<br>" +  
    Number("  10") + "<br>" +  
    Number("10  ") + "<br>" +  
    Number("10 6");  
< /script>  
  
< /body>  
< /html>  
  
  
The parseInt() Method  
parseInt() parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:  
  
Example  
  
parseInt("10");         // returns 10  
parseInt("10.33");      // returns 10  
parseInt("10 20 30");   // returns 10  
parseInt("10 years");   // returns 10  
parseInt("years 10");   // returns NaN  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The global JavaScript function parseInt() converts strings to numbers:</p>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML =  
    parseInt("10") + "<br>" +  
    parseInt("10.33") + "<br>" +  
    parseInt("10 6") + "<br>" +  
    parseInt("10 years") + "<br>" +  
    parseInt("years 10");  
< /script>  
  
< /body>  
< /html>  
  
  
If the number cannot be converted, NaN (Not a Number) is returned.  
  
The parseFloat() Method  
parseFloat() parses a string and returns a number. Spaces are allowed. Only the first number is returned:  
  
Example  
  
parseFloat("10");        // returns 10  
parseFloat("10.33");     // returns 10.33  
parseFloat("10 20 30");  // returns 10  
parseFloat("10 years");  // returns 10  
parseFloat("years 10");  // returns NaN  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The global JavaScript function parseFloat() converts strings to numbers:</p>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML =  
    parseFloat("10") + "<br>" +  
    parseFloat("10.33") + "<br>" +  
    parseFloat("10 6") + "<br>" +  
    parseFloat("10 years") + "<br>" +  
    parseFloat("years 10");  
< /script>  
  
< /body>  
< /html>  
  
  
  
If the number cannot be converted, NaN (Not a Number) is returned.  
  
The valueOf() Method  
valueOf() returns a number as a number.  
  
Example  
  
var x = 123;  
x.valueOf();            // returns 123 from variable x  
(123).valueOf();        // returns 123 from literal 123  
(100 + 23).valueOf();   // returns 123 from expression 100 + 23  
  
< !DOCTYPE html>  
< html>  
< body>  
< p id="demo"></p>  
  
< script>  
var x = 123;  
  
document.getElementById("demo").innerHTML =  
    x.valueOf() + "<br>" +  
    (123).valueOf() + "<br>" +  
    (100 + 23).valueOf();  
< /script>  
  
< /body>  
< /html>  
  
In JavaScript, a number can be a primitive value (typeof = number) or an object (typeof = object).  
  
The valueOf() method is used internally in JavaScript to convert Number objects to primitive values.  
  
There is no reason to use it in your code.  
  
Note    In JavaScript, all data types have a valueOf() and a toString() method.**

**J17**

**= is used to assign values, + is used to add values, ++ is used to increment values.  
  
The assignment operator = is used to assign values to JavaScript variables.  
  
The arithmetic operator + is used to add values together.  
  
Example  
  
Assign values to variables and add them together:  
  
y = 5;           // assign the value 5 to y  
z = 2;           // assign the value 2 to z  
x = y + z;       // assign the value 7 (y + z) to x  
The result of x will be:  
  
7  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>y = 5, z = 2, calculate x = y + z, and display x:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    y = 5;  
    z = 2;  
    x = y + z;  
    document.getElementById("demo").innerHTML = x;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
JavaScript Arithmetic Operators  
Arithmetic operators are used to perform arithmetic between variables and/or values.  
  
Given that y = 5, the table below explains the arithmetic operators:  
  
Operator    Description    Example    Result    Result    Try it  
+    Addition    x = y + 2    y = 5    x = 7    Try it »  
-    Subtraction    x = y - 2    y = 5    x = 3    Try it »  
\*    Multiplication    x = y \* 2    y = 5    x = 10    Try it »  
/    Division    x = y / 2    y = 5    x = 2.5    Try it »  
%    Modulus (division remainder)    x = y % 2    y = 5    x = 1    Try it »  
++    Increment    x = ++y    y = 6    x = 6    Try it »  
x = y++    y = 6    x = 5    Try it »  
--    Decrement    x = --y    y = 4    x = 4    Try it »  
x = y--    y = 4    x = 5    Try it »  
  
JavaScript Assignment Operators  
Assignment operators are used to assign values to JavaScript variables.  
  
Given that x = 10 and y = 5, the table below explains the assignment operators:  
  
Operator    Example    Same As    Result    Try it  
=    x = y    x = y    x = 5    Try it »  
+=    x += y    x = x + y    x = 15    Try it »  
-=    x -= y    x = x - y    x = 5    Try it »  
\*=    x \*= y    x = x \* y    x = 50    Try it »  
/=    x /= y    x = x / y    x = 2    Try it »  
%=    x %= y    x = x % y    x = 0    Try it »  
  
JavaScript String Operators  
The + operator can also be used to concatenate (add) strings.  
  
Note    When used on strings, the + operator is called the concatenation operator.  
Example  
  
To add two or more string variables together, use the + operator.  
  
txt1 = "What a very";  
txt2 = "nice day";  
txt3 = txt1 + txt2;  
The result of txt3 will be:  
  
What a verynice day  
  
Try it Yourself »  
To add a space between the two strings, insert a space into one of the strings:  
  
Example  
  
txt1 = "What a very ";  
txt2 = "nice day";  
txt3 = txt1 + txt2;  
The result of txt3 will be:  
  
What a very nice day  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The + operator concatenates (adds) strings.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var txt1 = "What a very";  
    var txt2 = "nice day";  
    document.getElementById("demo").innerHTML = txt1 + txt2;  
}  
< /script>  
  
< /body>  
< /html>  
  
or insert a space into the expression:  
  
Example  
  
txt1 = "What a very";  
txt2 = "nice day";  
txt3 = txt1 + " " + txt2;  
The result of txt3 will be:  
  
What a very nice day  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The + operator concatenates (adds) strings.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var txt1 = "What a very";  
    var txt2 = "nice day";  
    document.getElementById("demo").innerHTML = txt1 + " " + txt2;  
}  
< /script>  
  
< /body>  
< /html>  
  
The += operator can also be used to concatenate strings:  
  
Example  
  
txt1 = "What a very ";  
txt1 += "nice day";  
The result of txt1 will be:  
  
What a very nice day  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The assignment operator += can concatenate strings.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    txt1 = "What a very ";  
    txt1 += "nice day";  
    document.getElementById("demo").innerHTML = txt1;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Adding Strings and Numbers  
Adding two numbers, will return the sum, but adding a number and a string will return a string:  
  
Example  
  
x = 5 + 5;  
y = "5" + 5;  
z= "Hello" + 5;  
The result of x, y, and z will be:  
  
10  
55  
Hello5  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Adding a number and a string, returns a string.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var x = 5 + 5;  
    var y = "5" + 5;  
    var z = "Hello" + 5;  
  
    document.getElementById("demo").innerHTML =  
    x + "<br>" +  
    y + "<br>" +  
    z;  
}  
< /script>  
  
< /body>  
< /html>  
  
The rule is: If you add a number and a string, the result will be a string!  
  
JavaScript Bitwise Operators  
Bit operators work on 32 bits numbers.  
  
Any numeric operand in the operation is converted into a 32 bit number.  
  
The result is converted back to a JavaScript number.  
  
Example  
  
x = 5 & 1;  
The result in x:  
  
1  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The bitwise operator & returna a logical AND.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = 5 & 1;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Operator    Description    Example    Same as    Result    Decimal  
&     AND    x = 5 & 1    0101 & 0001    0001    1  
|    OR    x = 5 | 1    0101 | 0001    0101    5  
~    NOT    x = ~ 5     ~0101    1010    10  
^    XOR    x = 5 ^ 1    0101 ^ 0001    0100    4  
<<     Left shift    x = 5 << 1    0101 << 1    1010    10  
>>     Right shift    x = 5 >> 1    0101 >> 1    0010    2  
  
Note    The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers.  
  
Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.  
  
~00000000000000000000000000000101 will return 11111111111111111111111111111010  
The typeof operator  
The typeof operator returns the type of a variable (or an expression):  
  
Example  
  
var x = 5;  
var y = "John";  
typeof x                      // Returns number  
typeof y                      // Returns string  
typeof 3.14                   // Returns number  
typeof false                  // Returns boolean  
typeof [1,2,3,4]              // Returns object  
typeof {name:'john', age:34}  // Returns object  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The typeof operator returns the type of a variable or expression.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var x = 5;  
    var y = "John";  
    document.getElementById("demo").innerHTML =  
    typeof x + "<br>" +  
    typeof 5 + "<br>" +  
    typeof (5+5) + "<br>" +  
    typeof y + "<br>" +  
    typeof "5" + "<br>" +  
    typeof "John" + "<br>" +  
    typeof ("John" + " " + "Doe") + "<br>" +  
    typeof false + "<br>" +  
    typeof (5 > 3) + "<br>" +  
    typeof (3 > 5) + "<br>" +  
    typeof [1,2,3,4] + "<br>" +  
    typeof {name:'john', age:34};  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
  
The delete Operator  
The delete operator can be used to delete properties from objects:  
  
Example  
  
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
delete person.age;  
The delete operator is designed to be used on object properties. It has no effect on variables or functions.  
  
The delete operator should not be used on predefined JavaScript object properties. It can crash your application.  
  
The Unary + Operator  
The unary + operator can be used to convert a variable to a number:  
  
Example  
  
var y = "5";      // y is a string  
var x = + y;      // x is a number  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The typeof operator returns the type of a variable or expression.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var y = "5";  
    var x = + y;  
    document.getElementById("demo").innerHTML =  
    typeof y + "<br>" + typeof x;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
If the variable cannot be converted, it will still become a number, but with the value NaN (Not a number):  
  
Example  
  
var y = "John";   // y is a string  
var x = + y;      // x is a number (NaN)  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The typeof operator returns the type of a variable or expression.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var y = "John";  
    var x = + y;  
    document.getElementById("demo").innerHTML =  
    typeof x + "<br>" + x;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
JavaScript Comparison and Logical Operators  
Comparison and logical operators are described in the JS Comparisons chapter**

**J18**

**The Math object allows you to perform mathematical tasks on numbers.  
  
The Math Object  
The Math object allows you to perform mathematical tasks.  
  
The Math object includes several mathematical methods.  
  
One common use of the Math object is to create a random number:  
  
Example  
  
Math.random();       // returns a random number  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Math.random() returns a random number betwween 0 and 1.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = Math.random();  
}  
< /script>  
  
< /body>  
< /html>  
  
Note     Math has no constructor. No methods have to create a Math object first.  
Math.min() and Math.max()  
Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments:  
  
Example  
  
Math.min(0, 150, 30, 20, -8);     // returns -8  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Math.min() returns the lowest value.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML =  
    Math.min(0, 150, 30, 20, -8);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Example  
  
Math.max(0, 150, 30, 20, -8);     // returns 150  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Math.max() returns the higest value.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML =  
    Math.max(0, 150, 30, 20, -8);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Math.random()  
Math.random() returns a random number between 0 and 1:  
  
Example  
  
Math.random();                    // returns a random number  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Math.random() returns a random number betwween 0 and 1.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = Math.random();  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Math.round()  
Math.round() rounds a number to the nearest integer:  
  
Example  
  
Math.round(4.7);                  // returns 5  
Math.round(4.4);                   // returns 4  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Math.round() rounds a number to its nearest integer.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = Math.round(4.4);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Math.ceil()  
Math.ceil() rounds a number up to the nearest integer:  
  
Example  
  
Math.ceil(4.4);                  // returns 5  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Math.ceil() rounds a number <strong>up</strong> to its nearest integer.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = Math.ceil(4.4);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Math.floor()  
Math.floor() rounds a number down to the nearest integer:  
  
Example  
  
Math.floor(4.7);                  // returns 4  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Math.floor() rounds a number <strong>down</strong> to its nearest integer.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = Math.floor(4.7);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Math.floor() and Math.random() can be used together return a random number between 0 and 10:  
  
Example  
  
Math.floor(Math.random() \* 11);   // returns a random number between 0 and 10  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Math.floor() combined with Math.random() can return random integers.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML =  
    Math.floor(Math.random() \* 11);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Math Constants  
JavaScript provides 8 mathematical constants that can be accessed with the Math object:  
  
Example  
  
Math.E;         // returns Euler's number  
Math.PI         // returns PI  
Math.SQRT2      // returns the square root of 2  
Math.SQRT1\_2    // returns the square root of 1/2  
Math.LN2        // returns the natural logarithm of 2  
Math.LN10       // returns the natural logarithm of 10  
Math.LOG2E      // returns base 2 logarithm of E  
Math.LOG10E     // returns base 10 logarithm of E  
  
Try it yourself »  
  
Math Object Methods  
Method    Description  
abs(x)    Returns the absolute value of x  
acos(x)    Returns the arccosine of x, in radians  
asin(x)    Returns the arcsine of x, in radians  
atan(x)    Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians  
atan2(y,x)    Returns the arctangent of the quotient of its arguments  
ceil(x)    Returns x, rounded upwards to the nearest integer  
cos(x)    Returns the cosine of x (x is in radians)  
exp(x)    Returns the value of Ex  
floor(x)    Returns x, rounded downwards to the nearest integer  
log(x)    Returns the natural logarithm (base E) of x  
max(x,y,z,...,n)    Returns the number with the highest value  
min(x,y,z,...,n)    Returns the number with the lowest value  
pow(x,y)    Returns the value of x to the power of y  
random()    Returns a random number between 0 and 1  
round(x)    Rounds x to the nearest integer  
sin(x)    Returns the sine of x (x is in radians)  
sqrt(x)    Returns the square root of x  
tan(x)    Returns the tangent of an angle**

**J19**

**The Date object lets you work with dates (years, months, days, minutes, seconds, milliseconds)  
  
Displaying Dates  
In this tutorial we use a script to display dates inside a <p> element with id="demo":  
  
Example  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML = Date();  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML = Date();  
< /script>  
  
< /body>  
< /html>  
  
  
The script above says: assign the value of Date() to the content (innerHTML) of the element with id="demo".  
  
Note    You will learn how to display a date, in a more readable format, at the bottom of this page.  
  
Creating Date Objects  
The Date object lets us work with dates.  
  
A date consists of a year, a month, a week, a day, a minute, a second, and a millisecond.  
  
Date objects are created with the new Date() constructor.  
  
There are 4 ways of initiating a date:  
  
new Date()  
new Date(milliseconds)  
new Date(dateString)  
new Date(year, month, day, hours, minutes, seconds, milliseconds)  
Using new Date(), without parameters, creates a new date object with the current date and time:  
  
Example  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< /body>  
< /html>  
  
  
Using new Date(), with a date string, creates a new date object with the specified date and time:  
  
Example  
  
< script>  
var d = new Date("October 13, 2014 11:13:00");  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date("October 13, 2014 11:13:00");  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< /body>  
< /html>  
  
  
Using new Date(), with a number, creates a new date object with number of millisecond since 1970/01/01:  
  
Example  
  
< script>  
var d = new Date(0);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date(0);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< /body>  
< /html>  
  
  
Note    JavaScript dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC).  
One day contains 86,400,000 millisecond.  
Using new Date(), with 7 numbers, creates a new date object with the specified date and time:  
  
The 7 numbers specify the year, month, day, hour, minute, second, and millisecond, in that order:  
  
Example  
  
< script>  
var d = new Date(99,5,24,11,33,30,0);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date(99,5,24,11,33,30,0);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< /body>  
< /html>  
  
  
Variants of the example above let us omit any of the last 4 parameters:  
  
Example  
  
< script>  
var d = new Date(99,5,24);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date(99,5,24);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< /body>  
< /html>  
  
  
Note    JavaScript counts months from 0 to 11. January is 0. December is 11.  
Date Methods  
When a Date object is created, a number of methods allow you to operate on it.  
  
Date methods allow you to get and set the year, month, day, hour, minute, second, and millisecond of objects, using either local time or UTC (universal, or GMT) time.  
  
The next chapter, of this tutorial, covers the date object's methods.  
  
Displaying Dates  
When you display a date object in HTML, it is automatically converted to a string, with the toString() method.  
  
Example  
  
< p id="demo"></p>  
  
< script>  
d = new Date();  
document.getElementById("demo").innerHTML = d;  
< /script>  
Is the same as:  
  
< p id="demo"></p>  
  
< script>  
d = new Date();  
document.getElementById("demo").innerHTML = d.toString();  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.toString();  
< /script>  
  
< /body>  
< /html>  
  
  
The toUTCString() method converts a date to a UTC string (a date display standard).  
  
Example  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.toUTCString;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The toUTCString() method converts a date to a UTC string (date display  
standard).</p>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.toUTCString();  
< /script>  
  
< /body>  
< /html>  
  
  
The toDateString() method converts a date to a more readable format:  
  
Example  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.toDateString;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.toDateString();  
< /script>  
  
< /body>  
< /html>**

**J20**

**Date methods let you get and set date values (years, months, days, minutes, seconds, milliseconds)  
  
Date Get Methods  
Get methods are used for getting a part of a date. Here are the most common (alphabetically):  
  
Method    Description  
getDate()    Get the day as a number (1-31)  
getDay()    Get the weekday a number (0-6)  
getFullYear()    Get the four digit year (yyyy)  
getHours()    Get the hour (0-23)  
getMilliseconds()    Get the milliseconds (0-999)  
getMinutes()    Get the minutes (0-59)  
getMonth()    Get the month (0-11)  
getSeconds()    Get the seconds (0-59)  
getTime()    Get the time (milliseconds since January 1, 1970)  
  
The getTime() Method  
getTime() returns the the number of milliseconds since 01.01.1970:  
  
Example  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.getTime();  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The internal clock in JavaScript starts at midnight January 1, 1970.</p>  
< p>The getTime() function returns the number of millisecodns since then:</p>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.getTime();  
< /script>  
  
< /body>  
< /html>  
  
  
The getFullYear() Method  
getFullYear() returns the year of a date as a four digit number:  
  
Example  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.getFullYear();  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The getFullYear() method returns the full year of a date:</p>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.getFullYear();  
< /script>  
  
< /body>  
< /html>  
  
  
The getDay() Method  
getDay() returns the weekday as a number (0-6):  
  
Example  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.getDay();  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The getDay() function returns the weekday as a number:</p>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.getDay();  
< /script>  
  
< /body>  
< /html>  
  
  
You can use an array of names, and getDay() to return the weekday as a name:  
  
Example  
  
< script>  
var d = new Date();  
var days = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"];  
document.getElementById("demo").innerHTML = days[d.getDay()];  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>You can use an array to display the name of the weekday:</p>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date();  
var days = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"];  
document.getElementById("demo").innerHTML = days[d.getDay()];  
< /script>  
  
< /body>  
< /html>  
  
  
Date Set Methods  
Set methods are used for setting a part of a date. Here are the most common (alphabitically):  
  
Method    Description  
setDate()    Set the day as a number (1-31)  
setFullYear()    Set the year (optionally month and day yyyy.mm.dd)  
setHours()    Set the hour (0-23)  
setMilliseconds()    Set the milliseconds (0-999)  
setMinutes()    Set the minutes (0-59)  
setMonth()    Set the month (0-11)  
setSeconds()    Set the seconds (0-59)  
setTime()    Set the time (milliseconds since January 1, 1970)  
  
The setFullYear() Method  
setFullYear() sets a date object to a specific date. In this example, to January 14, 2020:  
  
Example  
  
< script>  
var d = new Date();  
d.setFullYear(2020, 0, 14);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The setFullYear() method sets a date object to a special date.</p>  
  
< p>Remember that JavaScript counts months from 0 to 11. Month 10 is November.</p>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date();  
d.setFullYear(2020, 0, 14);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< /body>  
< /html>  
  
  
The setDate() Method  
setDate() sets the day of the month (1-31):  
  
Example  
  
< script>  
var d = new Date();  
d.setDate(20);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The setDate() method sets the date of a month.</p>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date();  
d.setDate(15);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< /body>  
< /html>  
  
  
The setDate() method can also be used to add days to a date:  
  
Example  
  
< script>  
var d = new Date();  
d.setDate(d.getDate() + 50);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The setDate() method can be used to add days to a date.</p>  
  
< p id="demo"></p>  
  
< script>  
var d = new Date();  
d.setDate(d.getDate() + 50);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< /body>  
< /html>  
  
  
Note    If adding days, shifts the month or year, the changes are handled automatically by the Date object.  
Date  Input - Parsing Dates  
If you have an input value (or any string), you can use the Date.parse() method to convert it to milliseconds.  
  
Date.parse() returns the number of milliseconds between the date and January 1, 1970:  
  
Example  
  
< script>  
var msec = Date.parse("March 21, 2012");  
document.getElementById("demo").innerHTML = msec;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Date.parse() returns the number of milliseconds between the date and January 1, 1970:</p>  
  
< p id="demo"></p>  
  
< script>  
var msec = Date.parse("March 21, 2012");  
document.getElementById("demo").innerHTML = msec;  
< /script>  
  
< /body>  
< /html>  
  
  
You can then use the number of milliseconds to convert it to a date object:  
  
Example  
  
< script>  
var msec = Date.parse("March 21, 2012");  
var d = new Date(msec);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Date.parse(string) returns milliseconds.</p>  
< p>You can use the return value to convert the string to a date object:</p>  
  
< p id="demo"></p>  
  
< script>  
var msec = Date.parse("March 21, 2012");  
var d = new Date(msec);  
document.getElementById("demo").innerHTML = d;  
< /script>  
  
< /body>  
< /html>  
  
  
Compare Dates  
Dates can easily be compared.  
  
The following example compares today's date with January 14, 2100:  
  
Example  
  
var today, someday, text;  
today = new Date();  
someday = new Date();  
someday.setFullYear(2100, 0, 14);  
  
if (someday > today) {  
    text = "Today is before January 14, 2100.";  
} else {  
    text = "Today is after January 14, 2100.";  
}  
document.getElementById("demo").innerHTML = text;  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var today, someday, text;  
today = new Date();  
someday = new Date();  
someday.setFullYear(2100, 0, 14);  
  
if (someday > today) {  
    text = "Today is before January 14, 2100.";  
} else {  
    text = "Today is after January 14, 2100.";  
}  
document.getElementById("demo").innerHTML = text;  
< /script>  
  
< /body>  
< /html>  
  
Note    JavaScript counts months from 0 to 11. January is 0. December is 11.  
Complete JavaScript Date Reference  
For a complete reference, go to our Complete JavaScript Date Reference.  
  
The reference contains descriptions and examples of all Date properties and methods.**

**J21**

**JavaScript arrays are used to store multiple values in a single variable.  
  
Displaying Arrays  
In this tutorial we will use a script to display arrays inside a <p> element with id="demo":  
  
Example  
  
< p id="demo"></p>  
  
< script>  
var cars = ["Saab", "Volvo", "BMW"];  
document.getElementById("demo").innerHTML = cars;  
< /script>  
The first line (in the script) creates an array named cars.  
  
The second line "finds" the element with id="demo", and "displays" the array in the "innerHTML" of it.  
  
Try it Yourself  
Create an array, and assign values to it:  
  
Example  
  
var cars = ["Saab", "Volvo", "BMW"];  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var cars = ["Saab", "Volvo", "BMW"];  
document.getElementById("demo").innerHTML = cars[0];  
< /script>  
  
< /body>  
< /html>  
  
  
Spaces and line breaks are not important. A declaration can span multiple lines:  
  
Example  
  
var cars = [  
    "Saab",  
    "Volvo",  
    "BMW"  
];  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var cars = [  
    "Saab",  
    "Volvo",  
    "BMW"  
];  
document.getElementById("demo").innerHTML = cars[0];  
< /script>  
  
< /body>  
< /html>  
  
  
Note    Don't put a comma after the last element (like "BMW",). It is inconsistent across browsers.  
What is an Array?  
An array is a special variable, which can hold more than one value at a time.  
  
If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:  
  
var car1 = "Saab";  
var car2 = "Volvo";  
var car3 = "BMW";  
However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?  
  
The solution is an array!  
  
An array can hold many values under a single name, and you can access the values by referring to an index number.  
  
Creating an Array  
Using an array literal is the easiest way to create a JavaScript Array.  
  
Syntax:  
  
var array-name = [item1, item2, ...];  
Example:  
  
var cars = ["Saab", "Volvo", "BMW"];  
  
Using the JavaScript Keyword new  
The following example also creates an Array, and assigns values to it:  
  
Example  
  
var cars = new Array("Saab", "Volvo", "BMW");  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var cars = new Array("Saab", "Volvo", "BMW");  
document.getElementById("demo").innerHTML = cars[0];  
< /script>  
  
< /body>  
< /html>  
  
  
Note    The two examples above do exactly the same. There is no need to use new Array().  
For simplicity, readability and execution speed, use the first one (the array literal method).  
Access the Elements of an Array  
You refer to an array element by referring to the index number.  
  
This statement access the value of the first element in myCars:  
  
var name = cars[0];  
This statement modifies the first element in cars:  
  
cars[0] = "Opel";  
  
Note    [0] is the first element in an array. [1] is the second. Array indexes start with 0.  
  
You Can Have Different Objects in One Array  
JavaScript variables can be objects. Arrays are special kinds of objects.  
  
Because of this, you can have variables of different types in the same Array.  
  
You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:  
  
myArray[0] = Date.now;  
myArray[1] = myFunction;  
myArray[2] = myCars;  
  
Arrays are Objects  
Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.  
  
But, JavaScript arrays are best described as arrays.  
  
Arrays use numbers to access its "elements". In this example, person[0] returns John:  
  
Array:  
  
var person = ["John", "Doe", 46];  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var person = ["John", "Doe", 46];  
document.getElementById("demo").innerHTML = person[0];  
< /script>  
  
< /body>  
< /html>  
  
  
Objects use names to access its "members". In this example, person.firstName returns John:  
  
Object:  
  
var person = {firstName:"John", lastName:"Doe", age:46};  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var person = {firstName:"John", lastName:"Doe", age:46};  
document.getElementById("demo").innerHTML = person["firstName"];  
< /script>  
  
< /body>  
< /html>  
  
  
Array Properties and Methods  
The real strength of JavaScript arrays are the built-in array properties and methods:  
  
Examples  
  
var x = cars.length;         // The length property returns the number of elements in cars  
var y = cars.sort();         // The sort() method sort cars in alphabetical order  
Array methods are covered in the next chapter.  
  
The length Property  
The length property of an array returns the length of an array (the number of array elements).  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.length;                       // the length of fruits is 4  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The length property returns the length of an array.</p>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.length;  
< /script>  
  
< /body>  
< /html>  
  
Note    The length property is always one more than the highest array index.  
Adding Array Elements  
The easiest way to add a new element to an array is to use the length property:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[fruits.length] = "Lemon";     // adds a new element (Lemon) to fruits  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The length property provides an easy way to append new elements to an array without using the push() method.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
  
function myFunction() {  
    fruits[fruits.length] = "Kiwi";  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< /body>  
< /html>  
  
Adding elements with high indexes can create undefined "holes" in an array:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[10] = "Lemon";                // adds a new element (Lemon) to fruits  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The length property provides an easy way to append new elements to an array without using the push() method.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
  
function myFunction() {  
    fruits[fruits.length] = "Kiwi";  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Looping Array Elements  
The best way to loop through an array is using a standard for loop:  
  
Example  
  
var index;  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
for    (index = 0; index < fruits.length; index++) {  
    text += fruits[index];  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The best way to loop through an array is using a standard for loop:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var index;  
    var text = "<ul>";  
    var fruits = ["Banana", "Orange", "Apple", "Mango"];  
    for (index = 0; index < fruits.length; index++) {  
        text += "<li>" + fruits[index] + "</li>";  
    }  
    text += "</ul>";  
    document.getElementById("demo").innerHTML = text;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Associative Arrays? No Way!  
Many programming languages support arrays with named indexes.  
  
Arrays with named indexes are called associative arrays (or hashes).  
  
JavaScript does not support arrays with named indexes.  
  
 Wrong:  
  
var person = new Array()  
person["firstName"] = "John";  
person["lastName"] = "Doe";  
person["age"] = 46;  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo1"></p>  
< p id="demo2"></p>  
  
< script>  
var person = new Array();  
person["firstName"] = "John";  
person["lastName"] = "Doe";  
person["age"] = 46;  
document.getElementById("demo1").innerHTML = person["firstName"];  
document.getElementById("demo2").innerHTML = person[0];  
< /script>  
  
< /body>  
< /html>  
  
  
The example above looks like it works. But it does not.  
  
If you try it, person["firstName"] will return John, but person[0] will return undefined, and person.length will return 0.  
  
Note    If you want to create an associative array, create an object instead.  
When to Use Arrays? When to use Objects?  
JavaScript does not support associative arrays.  
You should use objects when you want the element names to be strings.  
You should use arrays when you want the element names to be sequential numbers.  
Avoid new Array()  
There is no need to use the JavaScript's built-in array constructor new Array().  
  
Use [] instead.  
  
These two different statements both create a new empty array named points:  
  
var points = new Array();         // Bad  
var points = [];                  // Good  
These two different statements both create a new array containing 6 numbers:  
  
var points = new Array(40, 100, 1, 5, 25, 10)  // Bad  
var points = [40, 100, 1, 5, 25, 10];          // Good  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Avoid using new Array(). Use [] instead.</p>  
  
< p id="demo"></p>  
  
< script>  
//var points = new Array(40, 100, 1, 5, 25, 10);  
var points = [40, 100, 1, 5, 25, 10];  
document.getElementById("demo").innerHTML = points[0];  
< /script>  
  
< /body>  
< /html>  
  
The new keyword complicates your code and produces nasty side effects:  
  
var points = new Array(40, 100);  // Creates an array with two elements (40 and 100)  
What if I remove one of the elements?  
  
var points = new Array(40);       // Creates an array with 40 undefined elements !!!!!  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Avoid using new Array().</p>  
  
< p id="demo"></p>  
  
< script>  
var points = new Array(40);  
document.getElementById("demo").innerHTML = points[0];  
< /script>  
  
< /body>  
< /html>  
  
  
How to Recognize an Array?  
A common question is: How do I know if a variable is an array?  
  
The problem is that the JavaScript operator typeof returns "object":  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
  
typeof fruits;             // typeof returns object  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The typeof operator, when used on an array, returns object.</p>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = typeof fruits;  
< /script>  
  
< /body>  
< /html>  
  
  
The typeof operator returns object because a JavaScript array is an object.  
  
To solve this problem you can create your own isArray() function:  
  
function isArray(myArray) {  
    return myArray.constructor.toString().indexOf("Array") > -1;  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>This "home made" function, when used on an array, returns true.</p>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = isArray(fruits);  
  
function isArray(myArray) {  
    return myArray.constructor.toString().indexOf("Array") > -1;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The function above always return true if the argument is an array.  
  
Or more precisely: it returns true if the object prototype of the argument is "[object array]".**

**J22**

**The strength of JavaScript arrays lies in the array methods.  
  
Converting Arrays to Strings  
In JavaScript, all objects have the valueOf() and toString() methods.  
  
The valueOf() method is the default behavior for an array. It returns an array as a string:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.valueOf();  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The valueOf() method returns an array as a comma separated string.</p>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.valueOf();  
< /script>  
  
< /body>  
< /html>  
  
  
For JavaScript arrays, valueOf() and toString() are equal.  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.toString();  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The toString() method returns an array as a comma separated string.</p>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.toString();  
< /script>  
  
< /body>  
< /html>  
  
The join() method also joins all array elements into a string.  
  
It behaves just like toString(), but you can specify the separator:  
  
Example  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange","Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.join(" \* ");  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>the join() method joins array elements into a string.</p>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.join(" \* ");  
< /script>  
  
< /body>  
< /html>  
  
  
Popping and Pushing  
When you work with arrays, it is easy to remove elements and add new elements.  
  
This is what popping and pushing is: Popping items out of an array, or pushing items into an array.  
  
The pop() method removes the last element from an array:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.pop();              // Removes the last element ("Mango") from fruits  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The pop method removes the last element from an array.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
  
function myFunction() {  
    fruits.pop()  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< /body>  
< /html>  
  
The push() method adds a new element to an array (at the end):  
  
Note    Remember: [0] is the first element in an array. [1] is the second. Array indexes start with 0.  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.push("Kiwi");       //  Adds a new element ("Kiwi") to fruits  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The push method appends a new element to an array.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
  
function myFunction() {  
    fruits.push("Kiwi")  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< /body>  
< /html>  
  
The pop() method returns the string that was "popped out".  
  
The push() method returns the new array length.  
  
Shifting Elements  
Shifting is equivalent to popping, working on the first element instead of the last.  
  
The shift() method removes the first element of an array, and "shifts" all other elements one place down.  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.shift();            // Removes the first element "Banana" from fruits  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The shift method removes (shifts) the first element of an array.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
  
function myFunction() {  
    fruits.shift();  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< /body>  
< /html>  
  
The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The unshift method adds new elements to the beginning of an array.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
  
function myFunction() {  
    fruits.unshift("Lemon");  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< p><b>Note:</b> The unshift() method does not work properly in Internet Explorer 8 and earlier, the values will be inserted, but the return value will be <em>undefined</em>.</p>  
  
< /body>  
< /html>  
  
  
The shift() method returns the string that was "shifted out".  
  
The unshift() method returns the new array length.  
  
Changing Elements  
Array elements are accessed using their index number:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[0] = "Kiwi";        // Changes the first element of fruits to "Kiwi"  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Array elements are accessed using their index number.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
  
function myFunction() {  
    fruits[0] = "Kiwi";  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The length property provides an easy way to append a new element to an array:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[fruits.length] = "Kiwi";          // Appends "Kiwi" to fruit  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The length property provides an easy way to append new elements to an array without using the push() method.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
  
function myFunction() {  
    fruits[fruits.length] = "Kiwi";  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Deleting Elements  
Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
delete fruits[0];           // Changes the first element in fruits to undefined  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Deleting an element leaves whole in an array.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
  
function myFunction() {  
    delete fruits[0];  
    document.getElementById("demo").innerHTML = fruits[0];  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Note    Using delete on array elements leaves undefined holes in the array. Use pop() or splice() instead.  
Splicing an Array  
The splice() method can be used to add new items to an array:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(2, 0, "Lemon", "Kiwi");  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The splice() method adds new elements to an array.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
function myFunction() {  
    fruits.splice(2, 0, "Lemon", "Kiwi");  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The first parameter (2) defines the position where new elements should be added (spliced in).  
  
The second parameter (0) defines how many elements should be removed.  
  
The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be added.  
  
Using splice() to Remove Elements  
With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(0,1);        // Removes the first element of fruits  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The splice() methods can be used to remove array elements.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
function myFunction() {  
    fruits.splice(0, 1);  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The first parameter (0) defines the position where new elements should be added (spliced in).  
  
The second parameter (1) defines how many elements should be removed.  
  
The rest of the parameters are omitted. No new elements will be added.  
  
Sorting an Array  
The sort() method sorts an array alphabetically:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();            // Sorts the elements of fruits  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The sort() method sorts an array alphabetically.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
  
function myFunction() {  
    fruits.sort();  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< /body>  
< /html>  
  
The sort() method takes a function as parameter. The function can be used to define the sort method.  
  
Reversing an Array  
The reverse() method reverses the elements in an array.  
  
You can use it to sort an array in descending order:  
  
Example  
  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();            // Sorts the elements of fruits  
fruits.reverse();         // Reverses the order of the elements  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The sort() method sorts an array alphabetically.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;  
  
function myFunction() {  
    fruits.sort();  
    fruits.reverse();  
    document.getElementById("demo").innerHTML = fruits;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Numeric Sort  
The sort() method cannot be used on a number array, because it sorts alphabetically (25 is bigger than 100).  
  
You can fix this by providing a function that returns -1, 0, or 1:  
  
Example  
  
var points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a-b});  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to sort the array in descending order.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var points = [40, 100, 1, 5, 25, 10];  
document.getElementById("demo").innerHTML = points;  
  
function myFunction() {  
    points.sort(function(a, b){return b-a});  
    document.getElementById("demo").innerHTML = points;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Use the same trick to sort an array descending:  
  
Example  
  
var points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return b-a});  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to sort the array in descending order.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
var points = [40, 100, 1, 5, 25, 10];  
document.getElementById("demo").innerHTML = points;  
  
function myFunction() {  
    points.sort(function(a, b){return b-a});  
    document.getElementById("demo").innerHTML = points;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Calling function(a, b) returns -1, 0, or 1, depending on the values of a and b.  
  
The arguments are provided by the sort() method when it compares two values.  
  
Example: When comparing 40 and 100, the sort() method calls function(40,100).  
  
Find the Highest (or Lowest) Value  
How to find the highest value in an array?  
  
Example  
  
var points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return b-a});  
// now points[0] contains the highest value  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The highest number is <span id="demo"></span></p>  
  
< script>  
var points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return b-a});  
document.getElementById("demo").innerHTML = points[0];  
< /script>  
  
< /body>  
< /html>  
  
  
And the lowest:  
  
Example  
  
var points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a-b});  
// now points[0] contains the lowest value  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The lowest number is <span id="demo"></span></p>  
  
< script>  
var points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a-b});  
document.getElementById("demo").innerHTML = points[0];  
< /script>  
  
< /body>  
< /html>  
  
  
  
Joining Arrays  
The concat() method creates a new array by concatenating two arrays:  
  
Example  
  
var myGirls = ["Cecilie", "Lone"];  
var myBoys = ["Emil", "Tobias","Linus"];  
var myChildren = myGirls.concat(myBoys);     // Concatenates (joins) myGirls and myBoys  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click "Try it" to display all my children:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var myGirls = ["Cecilie", "Lone"];  
    var myBoys = ["Emil", "Tobias", "Linus"];  
    var myChildren = myGirls.concat(myBoys);  
    document.getElementById("demo").innerHTML = myChildren;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The concat() method can take any number of array arguments:  
  
Example  
  
var arr1 = ["Cecilie", "Lone"];  
var arr2 = ["Emil", "Tobias","Linus"];  
var arr3 = ["Robin", "Morgan"];  
var myChildren = arr1.concat(arr2, arr3);     // Concatenates arr1 with arr2 and arr3  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to join three arrays.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var arr1 = ["Cecilie", "Lone"];  
    var arr2 = ["Emil", "Tobias", "Linus"];  
    var arr3 = ["Robin", "Morgan"];  
    document.getElementById("demo").innerHTML =  
    arr1.concat(arr2, arr3);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Slicing an Array  
The slice() method slices out a piece of an array:  
  
Example  
  
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
var citrus = fruits.slice(1,3);  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The slice() method slices elements from an array.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
    var citrus = fruits.slice(1,3);  
    document.getElementById("demo").innerHTML = citrus;  
}  
< /script>  
  
< /body>  
< /html>**

**J23**

**A JavaScript Boolean represents one of two values: true or false.  
  
The Boolean() Function  
You can use the Boolean() function to find out if an expression (or a variable) is true:  
  
Example  
  
Boolean(10 > 9)        // returns true  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Display the value of Boolean(10 > 9):</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = Boolean(10 > 9);  
}  
< /script>  
  
< /body>  
< /html>  
  
Or even easier:  
  
Example  
  
(10 > 9)              // also returns true  
10 > 9                // also returns true  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Display the value of 10 > 9:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
  
< script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = 10 > 9;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Note    The Boolean value of an expression is the fundament for JavaScript comparisons and conditions.  
  
Everything With a Real Value is True  
Examples  
  
100  
  
3.14  
  
-15  
  
"Hello"  
  
"false"  
  
7 + 1 + 3.14  
  
5 < 6  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
  
< script>  
var b1 = Boolean(100);  
var b2 = Boolean(3.14);  
var b3 = Boolean(-15);  
var b4 = Boolean("Hello");  
var b5 = Boolean('false');  
var b6 = Boolean(1 + 7 + 3.14);  
  
document.getElementById("demo").innerHTML =  
"100 is " + b1 + "<br>" +  
"3.14 is " + b2 + "<br>" +  
"-15 is " + b3 + "<br>" +  
"Any (not empty) string is " + b4 + "<br>" +  
"Even the string 'false' is " + b5 + "<br>" +  
"Any expression is (exept zero) is " + b6;  
< /script>  
  
< /body>  
< /html>  
  
  
Everything Without a Real Value is False  
The Boolean value of 0 (zero) is false:  
  
var x = 0;  
Boolean(x);       // returns false  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Display the Boolean value of 0:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
  
< script>  
function myFunction() {  
    var x = 0;  
    document.getElementById("demo").innerHTML = Boolean(x);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The Boolean value of -0 (minus zero) is false:  
  
var x = -0;  
Boolean(x);       // returns false  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Display the Boolean value of -0:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
  
< script>  
function myFunction() {  
    var x = -0;  
    document.getElementById("demo").innerHTML = Boolean(x);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The Boolean value of "" (empty string) is false:  
  
var x = "";  
Boolean(x);       // returns false  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Display the Boolean value of "":</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
  
< script>  
function myFunction() {  
    var x = "";  
    document.getElementById("demo").innerHTML = Boolean(x);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The Boolean value of undefined is false:  
  
var x;  
Boolean(x);       // returns false  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Display the Boolean value of undefined:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
  
< script>  
function myFunction() {  
    var x;  
    document.getElementById("demo").innerHTML = Boolean(x);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The Boolean value of null is false:  
  
var x = null;  
Boolean(x);       // returns false  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Display the Boolean value of null:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
  
< script>  
function myFunction() {  
    var x = null;  
    document.getElementById("demo").innerHTML = Boolean(x);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The Boolean value of false is (you guessed it) false:  
  
var x = false;  
Boolean(x);       // returns false  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Display the Boolean value of false:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
  
< script>  
function myFunction() {  
    var x = false;  
    document.getElementById("demo").innerHTML = Boolean(x);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The Boolean value of NaN is false:  
  
var x = 10 / "H";  
Boolean(x);       // returns false  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Display the Boolean value of NaN:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
  
< script>  
function myFunction() {  
    var x = 10 / "H";  
    document.getElementById("demo").innerHTML = Boolean(x);  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Boolean Properties and Methods  
Primitive values, like true and false, cannot have properties or methods (because they are not objects).  
  
But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.**

**J24**

**Number() converts to a Number, String() converts to a String, Boolean() converts to a Boolean.  
  
JavaScript Data Types  
In JavaScript there are 5 different data types that can contain values:  
  
string  
number  
boolean  
object  
function  
There are 3 types of objects:  
  
Object  
Date  
Array  
And 2 data types that cannot contain values:  
  
null  
undefined  
The typeof Operator  
You can use the typeof operator to find the data type of a JavaScript variable.  
  
Example  
  
typeof "John"                 // Returns string  
typeof 3.14                   // Returns number  
typeof NaN                    // Returns number  
typeof false                  // Returns boolean  
typeof [1,2,3,4]              // Returns object  
typeof {name:'John', age:34}  // Returns object  
typeof new Date()             // Returns object  
typeof function () {}         // Returns function  
typeof myCar                  // Returns undefined (if myCar is not declared)  
typeof null                   // Returns object  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The typeof operator returns the type of a variable, object, function or expression.</p>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML =  
    typeof "john" + "<br>" +  
    typeof 3.14 + "<br>" +  
    typeof NaN + "<br>" +  
    typeof false + "<br>" +  
    typeof [1,2,3,4] + "<br>" +  
    typeof {name:'john', age:34} + "<br>" +  
    typeof new Date() + "<br>" +  
    typeof function () {} + "<br>" +  
    typeof myCar + "<br>" +  
    typeof null;  
< /script>  
  
< /body>  
< /html>  
  
  
  
Please observe:  
  
The data type of NaN is number  
The data type of an array is object  
The data type of a date is object  
The data type of null is object  
The data type of an undefined variable is undefined  
You cannot use typeof to define if an object is an JavaScript Array or a JavaScript Date.  
  
The constructor Property  
The constructor property to returns the constructor function for all JavaScript variables.  
  
Example  
  
"John".constructor                 // Returns function String()  { [native code] }  
(3.14).constructor                 // Returns function Number()  { [native code] }  
false.constructor                  // Returns function Boolean() { [native code] }  
[1,2,3,4].constructor              // Returns function Array()   { [native code] }  
{name:'John', age:34}.constructor  // Returns function Object()  { [native code] }  
new Date().constructor             // Returns function Date()    { [native code] }  
function () {}.constructor         // Returns function Function(){ [native code] }  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The constructor property returns the constructor function for a variable or an  
object.</p>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML =  
    "john".constructor + "<br>" +  
    (3.14).constructor + "<br>" +  
    false.constructor + "<br>" +  
    [1,2,3,4].constructor + "<br>" +  
    {name:'john', age:34}.constructor + "<br>" +  
    new Date().constructor + "<br>" +  
    function () {}.constructor;  
< /script>  
  
< /body>  
< /html>  
  
  
  
You can check the constructor property to find out if an object is an Array (contains the word "Array"):  
  
Example  
  
function isArray(myArray) {  
    return myArray.constructor.toString().indexOf("Array") > -1;  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>This "home made" function, when used on an array, returns true.</p>  
  
< p id="demo"></p>  
  
< script>  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = isArray(fruits);  
  
function isArray(myArray) {  
    return myArray.constructor.toString().indexOf("Array") > -1;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
You can check the constructor property to find out if an object is a Date (contains the word "Date"):  
  
Example  
  
function isDate(myDate) {  
    return myDate.constructor.toString().indexOf("Date") > -1;  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>This "home made" function, when used on an date, returns true.</p>  
  
< p id="demo"></p>  
  
< script>  
var myDate = new Date();  
document.getElementById("demo").innerHTML = isDate(myDate);  
  
function isDate(myDate) {  
    return myDate.constructor.toString().indexOf("Date") > -1;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
JavaScript Type Conversion  
JavaScript variables can be converted to a new variable and another datatype:  
  
By the use of a JavaScript function  
Automatically by JavaScript itself  
Converting Numbers to Strings  
The global method String() can convert numbers to strings.  
  
It can be used on any type of numbers, literals, variables, or expressions:  
  
Example  
  
String(x)         // returns a string from a number variable x  
String(123)       // returns a string from a number literal 123  
String(100 + 23)  // returns a string from a number from an expression  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The String() method can convert a number to a string.</p>  
  
< p id="demo"></p>  
  
< script>  
var x = 123;  
document.getElementById("demo").innerHTML =  
    String(x) + "<br>" +  
    String(123) + "<br>" +  
    String(100 + 23);  
< /script>  
< /body>  
< /html>  
  
  
  
The Number method toString() does the same.  
  
Example  
  
x.toString()  
(123).toString()  
(100 + 23).toString()  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>The toString() method converts a number to a string.</p>  
  
< p id="demo"></p>  
  
< script>  
var x = 123;  
document.getElementById("demo").innerHTML =  
    x.toString() + "<br>" +  
   (123).toString() + "<br>" +  
   (100 + 23).toString();  
< /script>  
< /body>  
< /html>  
  
  
  
In the chapter Number Methods, you will find more methods that can be used to convert numbers to strings:  
  
Method    Description  
toExponential()    Returns a string, with a number rounded and written using exponential notation.  
toFixed()    Returns a string, with a number rounded and written with a specified number of decimals.  
toPrecision()    Returns a string, with a number written with a specified length  
  
Converting Booleans to Strings  
The global method String() can convert booleans to strings.  
  
String(false)        // returns "false"  
String(true)         // returns "true"  
The Boolean method toString() does the same.  
  
false.toString()     // returns "false"  
true.toString()      // returns "true"  
  
Converting Dates to Strings  
The global method String() can convert dates to strings.  
  
String(Date())      // returns Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)  
The Date method toString() does the same.  
  
Example  
  
Date().toString()   // returns Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)  
In the chapter Date Methods, you will find more methods that can be used to convert dates to strings:  
  
Method    Description  
getDate()    Get the day as a number (1-31)  
getDay()    Get the weekday a number (0-6)  
getFullYear()    Get the four digit year (yyyy)  
getHours()    Get the hour (0-23)  
getMilliseconds()    Get the milliseconds (0-999)  
getMinutes()    Get the minutes (0-59)  
getMonth()    Get the month (0-11)  
getSeconds()    Get the seconds (0-59)  
getTime()    Get the time (milliseconds since January 1, 1970)  
  
Converting Strings to Numbers  
The global method Number() can convert strings to numbers.  
  
Strings containing numbers (like "3.14") convert to numbers (like 3.14).  
  
Empty strings convert to 0.  
  
Anything else converts to NaN (Not a number).  
  
Number("3.14")    // returns 3.14  
Number(" ")       // returns 0  
Number("")        // returns 0  
Number("99 88")   // returns NaN  
In the chapter Number Methods, you will find more methods that can be used to convert strings to numbers:  
  
Method    Description  
parseFloat()    Parses a string and returns a floating point number  
parseInt()    Parses a string and returns an integer  
  
Converting Booleans to Numbers  
The global method Number() can also convert booleans to numbers.  
  
Number(false)     // returns 0  
Number(true)      // returns 1  
  
Converting Dates to Numbers  
The global method Number() can be used to convert dates to numbers.  
  
d = new Date();  
Number(d)          // returns 1404568027739  
The date method getTime() does the same.  
  
d = new Date();  
d.getTime()        // returns 1404568027739  
  
Automatic String Conversion  
JavaScript automatically calls the variable's toString() function when you try to output an object or a variable:  
  
document.GetElementById("demo").innerHTML = myVar  
  
// if myVar = {name:"Fjohn"}  // toString converts to "[object Object]"  
// if myVar = [1,2,3,4]       // toString converts to "1,2,3,4"  
// if myVar = new Date()      // toString converts to "Fri Jul 18 2014 09:08:55 GMT+0200"  
Numbers and booleans are also converted, but this is not very visible:  
  
// if myVar = 123             // toString converts to "123"  
// if myVar = true            // toString converts to "true"  
// if myVar = false           // toString converts to "false"**

**J25**

**Comparison and Logical operators are used to test for true or false.  
  
Comparison Operators  
Comparison operators are used in logical statements to determine equality or difference between variables or values.  
  
Given that x=5, the table below explains the comparison operators:  
  
Operator    Description    Comparing    Returns  
==    equal to    x == 8    false  
x == 5    true  
===    equal value and equal type    x === "5"    false  
x === 5    true  
!=    not equal    x != 8    true  
!==    not equal value or not equal type    x !== "5"    true  
x !== 5    false  
>      greater than    x > 8    false  
<      less than    x < 8    true  
> =     greater than or equal to    x >= 8    false  
< =     less than or equal to    x <= 8    true  
  
How Can it be Used  
Comparison operators can be used in conditional statements to compare values and take action depending on the result:  
  
if (age < 18) text = "Too young";  
You will learn more about the use of conditional statements in the next chapter of this tutorial.  
  
Logical Operators  
Logical operators are used to determine the logic between variables or values.  
  
Given that x=6 and y=3, the table below explains the logical operators:  
  
Operator    Description    Example  
&&     and    (x < 10 && y > 1) is true  
||    or    (x == 5 || y == 5) is false  
!    not    !(x == y) is true  
  
Conditional Operator  
JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.  
  
Syntax  
  
variablename = (condition) ? value1:value2  
Example  
  
voteable = (age < 18) ? "Too young":"Old enough";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Input your age and click the button:</p>  
  
< input id="age" value="18" />  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var age,voteable;  
    age = document.getElementById("age").value;  
    voteable = (age < 18) ? "Too young":"Old enough";  
    document.getElementById("demo").innerHTML = voteable + " to vote.";  
}  
< /script>  
  
< /body>  
< /html>  
  
If the variable age is a value below 18, the value of the variable voteable will be "Too young", otherwise the value of voteable will be "Old enough":**

**J26**

**Conditional statements are used to perform different actions based on different conditions.  
  
Conditional Statements  
Very often when you write code, you want to perform different actions for different decisions.  
  
You can use conditional statements in your code to do this.  
  
In JavaScript we have the following conditional statements:  
  
Use if to specify a block of code to be executed, if a specified condition is true  
Use else to specify a block of code to be executed, if the same condition is false  
Use else if to specify a new condition to test, if the first condition is false  
Use switch to specify many alternative blocks of code to be executed  
The if Statement  
Use the if statement to specify a block of JavaScript code to be executed if a condition is true.  
  
Syntax  
  
if (condition) {  
    block of code to be executed if the condition is true  
}  
  
Note    Note that if is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.  
  
Example  
  
Make a "Good day" greeting if the time is less than 20:00:  
  
if (time < 20) {  
    greeting = "Good day";  
}  
The result of greeting will be:  
  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Display "Good day", only if the time is less than 20:00:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    if (new Date().getHours() < 20) {  
        document.getElementById("demo").innerHTML = "Good day";  
    }  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The else Statement  
Use the else statement to specify a block of code to be executed if the condition is false.  
  
if (condition) {  
    block of code to be executed if the condition is true  
} else {  
    block of code to be executed if the condition is false  
}  
  
Example  
  
If the time is less than 20:00, create a "Good day" greeting, otherwise "Good evening":  
  
if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}  
The result of greeting will be:  
  
Good evening  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to display a time-based greeting:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var greeting;  
    if (new Date().getHours() < 20) {  
        greeting = "Good day";  
    } else {  
        greeting = "Good evening";  
    }  
    document.getElementById("demo").innerHTML = greeting;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The else if Statement  
Use the else if statement to specify a new condition if the first condition is false.  
  
Syntax  
  
if (condition1) {  
    block of code to be executed if condition1 is true  
} else if (condition2) {  
    block of code to be executed if the condition1 is false and condition2 is true  
} else {  
    block of code to be executed if the condition1 is false and condition2 is false  
}  
  
Example  
  
If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":  
  
if (time < 10) {  
    greeting = "Good morning";  
} else if (time<20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}  
The result of x will be:  
  
Good evening  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to get a time-based greeting:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var greeting;  
    var time = new Date().getHours();  
    if (time < 10) {  
        greeting = "Good morning";  
    } else if (time < 20) {  
        greeting = "Good day";  
    } else {  
        greeting ="Good evening";  
    }  
document.getElementById("demo").innerHTML = greeting;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Examples  
More Examples  
Random link  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
if (Math.random() > 0.5) {  
    text = "<a href='**[**http://w3schools.com**](http://w3schools.com/)**'>Visit W3Schools</a>";  
} else {  
    text = "<a href='**[**http://wwf.org**](http://wwf.org/)**'>Visit WWF</a>";  
}  
document.getElementById("demo").innerHTML = text;  
< /script>  
  
< /body>  
< /html>  
  
This example will write a link to either W3Schools or to the World Wildlife Foundation (WWF). By using a random number, there is a 50% chance for each of the links.**

**J27**

**The switch statement is used to perform different action based on different conditions.  
  
The JavaScript Switch Statement  
Use the switch statement to select one of many blocks of code to be executed.  
  
Syntax  
  
switch(expression) {  
    case n:  
        code block  
        break;  
    case n:  
        code block  
        break;  
    default:  
        default code block  
}  
This is how it works:  
  
The switch expression is evaluated once.  
The value of the expression is compared with the values of each case.  
If there is a match, the associated block of code is executed.  
  
Example  
  
Use today's weekday number to calculate weekday name: (Sunday=0, Monday=1, Tuesday=2, ...)  
  
switch (new Date().getDay()) {  
    case 0:  
        day = "Sunday";  
        break;  
    case 1:  
        day = "Monday";  
        break;  
    case 2:  
        day = "Tuesday";  
        break;  
    case 3:  
        day = "Wednesday";  
        break;  
    case 4:  
        day = "Thursday";  
        break;  
    case 5:  
        day = "Friday";  
        break;  
    case 6:  
        day = "Saturday";  
        break;  
}  
The result of day will be:  
  
Saturday  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to display what day it is today:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var day;  
    switch (new Date().getDay()) {  
        case 0:  
            day = "Sunday";  
            break;  
        case 1:  
            day = "Monday";  
            break;  
        case 2:  
            day = "Tuesday";  
            break;  
        case 3:  
            day = "Wednesday";  
            break;  
        case 4:  
            day = "Thursday";  
            break;  
        case 5:  
            day = "Friday";  
            break;  
        case  6:  
            day = "Saturday";  
            break;  
    }  
    document.getElementById("demo").innerHTML = "Today is " + day;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The break Keyword  
When the JavaScript code interpreter reaches a break keyword, it breaks out of the switch block.  
  
This will stop the execution of more execution of code and/or case testing inside the block.  
  
Note    When a match is found, and the job is done, it's time for a break.  
There is no need for more testing.  
  
The default Keyword  
The default keyword specifies the code to run if there is no case match:  
  
Example  
  
If today is neither Saturday nor Sunday, write a default message:  
  
switch (new Date().getDay()) {  
    case 6:  
        text = "Today is Saturday";  
        break;  
    case 0:  
        text = "Today is Sunday";  
        break;  
    default:  
        text = "Looking forward to the Weekend";  
}  
The result of text will be:  
  
Today is Saturday  
  
< html>  
< body>  
  
< p>Click the button to display a message based on what day it is today:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var text;  
    switch (new Date().getDay()) {  
        case 6:  
            text = "Today is Saturday";  
            break;  
        case 0:  
            text = "Today is Sunday";  
            break;  
        default:  
            text = "Looking forward to the Weekend";  
    }  
    document.getElementById("demo").innerHTML = text;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Common Code and Fall-Through  
Sometimes, in a switch block, you will want different cases to use the same code, or fall-through to a common default.  
  
Note from the next example, that cases can share the same code block, and that the default case does not have to be the last case in a switch block:  
  
Example  
  
switch (new Date().getDay()) {  
    case 1:  
    case 2:  
    case 3:  
    default:  
        text = "Looking forward to the Weekend";  
        break;  
    case 4:  
    case 5:  
        text = "Soon it is Weekend";  
        break;  
    case 0:  
    case 6:  
        text = "It is Weekend";  
}  
  
< html>  
< body>  
  
< p>Click the button to display a message based on what day it is:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var text;  
    switch (new Date().getDay()) {  
        case 1:  
        case 2:  
        case 3:  
        default:  
            text = "Looking forward to the Weekend";  
            break;  
        case 4:  
        case 5:  
            text = "Soon it is Weekend";  
            break;  
        case 0:  
        case 6:  
            text = "It is Weekend";  
    }  
    document.getElementById("demo").innerHTML = text;  
}  
< /script>  
  
< /body>  
< /html>**

**J28**

**Loops can execute a block of code a number of times.  
  
JavaScript Loops  
Loops are handy, if you want to run the same code over and over again, each time with a different value.  
  
Often this is the case when working with arrays:  
  
Instead of writing:  
  
text += cars[0] + "<br>";  
text += cars[1] + "<br>";  
text += cars[2] + "<br>";  
text += cars[3] + "<br>";  
text += cars[4] + "<br>";  
text += cars[5] + "<br>";  
  
You can write:  
  
for (i = 0; i < cars.length; i++) {  
    text += cars[i] + "<br>";  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
cars = ["BMW", "Volvo", "Saab", "Ford"];  
text = "";  
var i;  
for (i = 0; i < cars.length; i++) {  
    text += cars[i] + "<br>";  
}  
  
document.getElementById("demo").innerHTML = text;  
< /script>  
  
< /body>  
< /html>  
  
  
Different Kinds of Loops  
JavaScript supports different kinds of loops:  
  
for - loops through a block of code a number of times  
for/in - loops through the properties of an object  
while - loops through a block of code while a specified condition is true  
do/while - also loops through a block of code while a specified condition is true  
The For Loop  
The for loop is often the tool you will use when you want to create a loop.  
  
The for loop has the following syntax:  
  
for (statement 1; statement 2; statement 3) {  
    code block to be executed  
}  
Statement 1 is executed before the loop (the code block) starts.  
  
Statement 2 defines the condition for running the loop (the code block).  
  
Statement 3 is executed each time after the loop (the code block) has been executed.  
  
Example  
  
for (i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>";  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to loop through a block of code five times.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var text = "";  
    var i;  
    for (i = 0; i < 5; i++) {  
        text += "The number is " + i + "<br>";  
    }  
    document.getElementById("demo").innerHTML = text;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
From the example above, you can read:  
  
Statement 1 sets a variable before the loop starts (var i = 0).  
  
Statement 2 defines the condition for the loop to run (i must be less than 5).  
  
Statement 3 increases a value (i++) each time the code block in the loop has been executed.  
  
Statement 1  
Normally you will use statement 1 to initiate the variable used in the loop (var i = 0).  
  
This is not always the case, JavaScript doesn't care. Statement 1 is optional.  
  
You can initiate many values in statement 1 (separated by comma):  
  
Example:  
  
for (i = 0, len = cars.length, text = ""; i < len; i++) {  
    text += cars[i] + "<br>";  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
cars = ["BMW", "Volvo", "Saab", "Ford"];  
var i;  
for (i = 0, l = cars.length, text = ""; i < l; i++) {  
    text += cars[i] + "<br>";  
}  
  
document.getElementById("demo").innerHTML = text;  
< /script>  
  
< /body>  
< /html>  
  
  
And you can omit statement 1 (like when your values are set before the loop starts):  
  
Example:  
  
var i = 2;  
var len = cars.length;  
var text = "";  
for (; i < len; i++) {  
    text += cars[i] + "<br>";  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
cars = ["BMW", "Volvo", "Saab", "Ford"];  
var i = 2;  
var len = cars.length;  
var text = "";  
  
for (; i < len; i++) {  
    text += cars[i] + "<br>";  
}  
  
document.getElementById("demo").innerHTML = text;  
< /script>  
  
< /body>  
< /html>  
  
  
Statement 2  
Often statement 2 is used to evaluate the condition of the initial variable.  
  
This is not always the case, JavaScript doesn't care. Statement 2 is also optional.  
  
If statement 2 returns true, the loop will start over again, if it returns false, the loop will end.  
  
Note    If you omit statement 2, you must provide a break inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.  
  
Statement 3  
Often statement 3 increases the initial variable.  
  
This is not always the case, JavaScript doesn't care, and statement 3 is optional.  
  
Statement 3 can do anything like negative increment (i--), or larger increment (i = i + 15), or anything else.  
  
Statement 3 can also be omitted (like when you increment your values inside the loop):  
  
Example:  
  
var i = 0;  
len = cars.length;  
for (; i < len; ) {  
    text += cars[i] + "<br>";  
    i++;  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
cars = ["BMW","Volvo","Saab","Ford"];  
  
var i =0;  
var len = cars.length;  
var text = "";  
  
for (; i < len; ) {  
    text += cars[i] + "<br>";  
    i++;  
}  
document.getElementById("demo").innerHTML = text;  
< /script>  
  
< /body>  
< /html>  
  
  
The For/In Loop  
The JavaScript for/in statement loops through the properties of an object:  
  
Example  
  
var person = {fname:"John", lname:"Doe", age:25};  
  
var text = "";  
var x;  
for (x in person) {  
    text += person[x];  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var txt = "";  
var person = {fname:"John", lname:"Doe", age:25};  
var x;  
for (x in person) {  
    txt += person[x] + " ";  
}  
document.getElementById("demo").innerHTML = txt;  
< /script>  
  
< /body>  
< /html>  
  
  
The While Loop  
The while loop and the do/while loop will be explained in the next chapter.**

**J29**

**Loops can execute a block of code as long as a specified condition is true.  
  
The While Loop  
The while loop loops through a block of code as long as a specified condition is true.  
  
Syntax  
  
while (condition) {  
    code block to be executed  
}  
Example  
  
In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:  
  
Example  
  
while (i < 10) {  
    text += "The number is " + i;  
    i++;  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to loop through a block of as long as i is less than 10.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var text = "";  
    var i = 0;  
    while (i < 10) {  
        text += "<br>The number is " + i;  
        i++;  
    }  
    document.getElementById("demo").innerHTML = text;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Note    If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.  
  
The Do/While Loop  
The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.  
  
Syntax  
  
do {  
    code block to be executed  
}  
while (condition);  
Example  
  
The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:  
  
Example  
  
do {  
    text += "The number is " + i;  
    i++;  
}  
while (i < 10);  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to loop through a block of code as long as i is less than 10.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var text = ""  
    var i = 0;  
    do {  
        text += "<br>The number is " + i;  
        i++;  
    }  
    while (i < 10)  
    document.getElementById("demo").innerHTML = text;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Do not forget to increase the variable used in the condition, otherwise the loop will never end!  
  
Comparing For and While  
If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.  
  
The loop in this example uses a for loop to collect the car names from the cars array:  
  
Example  
  
cars = ["BMW","Volvo","Saab","Ford"];  
var i = 0;  
var text = "";  
  
for (;cars[i];) {  
    text += cars[i] + "<br>";  
    i++;  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
cars = ["BMW","Volvo","Saab","Ford"];  
var i = 0;  
var text = "";  
for (;cars[i];) {  
    text += cars[i] + "<br>";  
    i++;  
}  
document.getElementById("demo").innerHTML = text;  
< /script>  
  
< /body>  
< /html>  
  
  
The loop in this example uses a while loop to collect the car names from the cars array:  
  
Example  
  
cars = ["BMW","Volvo","Saab","Ford"];  
var i = 0;  
var text = "";  
  
while (cars[i]) {  
    text += cars[i] + "<br>";  
    i++;  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
cars = ["BMW","Volvo","Saab","Ford"];  
var i = 0;  
var text = "";  
while (cars[i]) {  
    text += cars[i] + "<br>";  
    i++;  
}  
document.getElementById("demo").innerHTML = text;  
< /script>  
  
< /body>  
< /html>**

**J30**

**The break statement "jumps out" of a loop.  
  
The continue statement "jumps over" one iteration in the loop.  
  
The Break Statement  
You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch() statement.  
  
The break statement can also be used to jump out of a loop.  
  
The break statement breaks the loop and continues executing the code after the loop (if any):  
  
Example  
  
for (i = 0; i < 10; i++) {  
    if (i == 3) {break;}  
    text += "The number is " + i + "<br>";  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to do a loop with a break.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var text = ""  
    var i;  
    for (i = 0; i < 10; i++) {  
        if (i == 3) {break;}  
        text += "The number is " + i + "<br>";  
    }  
    document.getElementById("demo").innerHTML = text;  
}  
< /script>  
  
< /body>  
< /html>  
  
Since the if statement has only one single line of code, the braces can be omitted:  
  
for (i = 0; i < 10; i++) {  
    if (i == 3) break;  
    text += "The number is " + i + "<br>";  
}  
  
The Continue Statement  
The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.  
  
This example skips the value of 3:  
  
Example  
  
for (i = 0; i <= 10; i++) {  
    if (i == 3) continue;  
    text += "The number is " + i + "<br>";  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Click the button to do a loop which will skip the step where i = 3.</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var text = "";  
    var i;  
    for (i = 0; i < 10; i++) {  
        if (i == 3) continue;  
        text += "The number is " + i + "<br>";  
    }  
    document.getElementById("demo").innerHTML = text;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
JavaScript Labels  
As you have already seen, in the chapter about the switch statement, JavaScript statements can be labeled.  
  
To label JavaScript statements you precede the statements with a label name and a colon:  
  
label:  
statements  
The break and the continue statements are the only JavaScript statements that can "jump out of" a code block.  
  
Syntax:  
  
break labelname;  
  
continue labelname;  
The continue statement (with or without a label reference) can only be used inside a loop.  
  
The break statement, without a label reference, can only be used inside a loop or a switch.  
  
With a label reference, it can be used to "jump out of" any JavaScript code block:  
  
Example  
  
cars = ["BMW", "Volvo", "Saab", "Ford"];  
list: {  
    text += cars[0] + "<br>";  
    text += cars[1] + "<br>";  
    text += cars[2] + "<br>";  
    text += cars[3] + "<br>";  
    break list;  
    text += cars[4] + "<br>";  
    text += cars[5] + "<br>";  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
cars = ["BMW", "Volvo", "Saab", "Ford"];  
text = "";  
  
list: {  
    text += cars[0] + "<br>";  
    text += cars[1] + "<br>";  
    text += cars[2] + "<br>";  
    break list;  
    text += cars[3] + "<br>";  
    text += cars[4] + "<br>";  
    text += cars[5] + "<br>";  
}  
  
document.getElementById("demo").innerHTML = text;  
< /script>  
  
< /body>  
< /html>**

**J31**

**The try statement lets you test a block of code for errors.  
  
The catch statement lets you handle the error.  
  
The throw statement lets you create custom errors.  
  
The finally statement lets you execute code, after try and catch, regardless of the result.  
  
Errors Will Happen!  
When executing JavaScript code, different errors can occur.  
  
Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things:  
  
Example  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
try {  
    adddlert("Welcome guest!");  
}  
catch(err) {  
    document.getElementById("demo").innerHTML = err.message;  
}  
< /script>  
  
< /body>  
< /html>  
  
In the example above we have made a typo in the code (in the try block).  
  
The catch block catches the error, and executes code to handle it:  
  
JavaScript try and catch  
The try statement allows you to define a block of code to be tested for errors while it is being executed.  
  
The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.  
  
The JavaScript statements try and catch come in pairs:  
  
try {  
    Block of code to try  
}  
catch(err) {  
    Block of code to handle errors  
}  
  
JavaScript Throws Errors  
When an error occurs, JavaScript will normally stop, and generate an error message.  
  
The technical term for this is: JavaScript will throw an error.  
  
The throw Statement  
The throw statement allows you to create a custom error.  
  
The technical term for this is: throw an exception.  
  
The exception can be a JavaScript String, a Number, a Boolean or an Object:  
  
throw "Too big";    // throw a text  
throw 500;          // throw a number  
If you use throw together with try and catch, you can control program flow and generate custom error messages.  
  
Input Validation Example  
This example examines input. If the value is wrong, an exception (err) is thrown.  
  
The exception (err) is caught by the catch statement and a custom error message is displayed:  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Please input a number between 5 and 10:</p>  
  
< input id="demo" type="text">  
< button type="button" onclick="myFunction()">Test Input</button>  
< p id="message"></p>  
  
< script>  
function myFunction() {  
    var message, x  
    message = document.getElementById("message");  
    message.innerHTML = "";  
    x = document.getElementById("demo").value;  
    try {  
        if(x == "") throw "is Empty";  
        if(isNaN(x)) throw "not a number";  
        if(x > 10) throw "too high";  
        if(x < 5) throw "too low";  
    }  
    catch(err) {  
        message.innerHTML = "Input " + err;  
    }  
}  
< /script>  
  
< /body>  
< /html>  
  
  
The finally Statement  
The finally statement lets you execute code, after try and catch, regardless of the result:  
  
try {  
    Block of code to try  
}  
catch(err) {  
    Block of code to handle errors  
}  
finally {  
    Block of code to be executed regardless of the try / catch result  
}  
  
Example  
  
function myFunction()  
    var message, x;  
    message = document.getElementById("message");  
    message.innerHTML = "";  
    x = document.getElementById("demo").value;  
    try {  
        if(x == "") throw "Empty";  
        if(isNaN(x)) throw "Not a number";  
        if(x > 10) throw "Too high";  
        if(x < 5) throw "Too low";  
    }  
    catch(err) {  
        message.innerHTML = "Error: " + err + ".";  
    }  
    finally {  
        document.getElementById("demo").value = "";  
    }  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Please input a number between 5 and 10:</p>  
  
< input id="demo" type="text">  
< button type="button" onclick="myFunction()">Test Input</button>  
  
< p id="message"></p>  
  
< script>  
function myFunction() {  
    var message, x;  
    message = document.getElementById("message");  
    message.innerHTML = "";  
    x = document.getElementById("demo").value;  
    try {  
        if(x == "")  throw "is Empty";  
        if(isNaN(x)) throw "not a number";  
        if(x > 10)   throw "too high";  
        if(x < 5)    throw "too low";  
    }  
    catch(err) {  
        message.innerHTML = "Input " + err;  
    }  
    finally {  
        document.getElementById("demo").value = "";  
    }  
}  
< /script>  
  
< /body>  
< /html>**

**J32**

**A regular expression is a sequence of characters that forms a search pattern.  
  
The search pattern can be used for text search and text replace operations.  
  
What Is a Regular Expression?  
A regular expression is a sequence of characters that forms a search pattern.  
  
When you search for data in a text, you can use this search pattern to describe what you are searching for.  
  
A regular expression can be a single character, or a more complicated pattern.  
  
Regular expressions can be used to perform all types of text search and text replace operations.  
  
Syntax  
/pattern/modifiers;  
  
Example:  
  
var patt = /w3schools/i  
Example explained:  
  
/w3schools/i  is a regular expression.  
  
w3schools  is a pattern (to be used in a search).  
  
i  is a modifier (modifies the search to be case-insensitive).  
  
Using String Methods  
In JavaScript, regular expressions are often used with the two string methods: search() and replace().  
  
The search() method uses an expression to search for a match, and returns the position of the match.  
  
The replace() method returns a modified string where the pattern is replaced.  
  
Using String search() With a Regular Expression  
Example  
  
Use a regular expression to do a case-insensitive search for "w3schools" in a string:  
  
var str = "Visit W3Schools";  
var n = str.search(/w3schools/i);  
The result in n will be:  
  
6  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Search a string for "w3Schools", and display the position of the match:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var str = "Visit W3Schools!";  
    var n = str.search(/w3Schools/i);  
    document.getElementById("demo").innerHTML = n;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
Using String search() With String  
The search method will also accept a string as search argument. The string argument will be converted to a regular expression:  
  
Example  
  
Use a string to do a search for "W3schools" in a string:  
  
var str = "Visit W3Schools!";  
var n = str.search("W3Schools");  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Search a string for "W3Schools", and display the position of the match:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    var str = "Visit W3Schools!";  
    var n = str.search("W3Schools");  
    document.getElementById("demo").innerHTML = n;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
  
Use String replace() With a Regular Expression  
Example  
  
Use a case insensitive regular expression to replace Microsoft with W3Schools in a string:  
  
var    str = "Visit Microsoft!";  
var res = str.replace(/microsoft/i, "W3Schools");  
The result in res will be:  
  
Visit W3Schools!  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Replace "microsoft" with "W3Schools" in the paragraph below:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo">Please visit Microsoft!</p>  
  
< script>  
function myFunction() {  
    var str = document.getElementById("demo").innerHTML;  
    var txt = str.replace(/microsoft/i,"W3Schools");  
    document.getElementById("demo").innerHTML = txt;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Using String replace() With a String  
The replace() method will also accept a string as search argument:  
  
var str = "Visit Microsoft!";  
var res = str.replace("Microsoft", "W3Schools");  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Replace "Microsoft" with "W3Schools" in the paragraph below:</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo">Please visit Microsoft!</p>  
  
< script>  
function myFunction() {  
    var str = document.getElementById("demo").innerHTML;  
    var txt = str.replace("Microsoft","W3Schools");  
    document.getElementById("demo").innerHTML = txt;  
}  
< /script>  
  
< /body>  
< /html>  
  
  
Did You Notice?  
  
Note    Regular expression arguments (instead of string arguments) can be used in the methods above.  
Regular expressions can make your search much more powerful (case insensitive for example).  
  
Regular Expression Modifiers  
Modifiers can be used to perform case-insensitive more global searches:  
  
Modifier    Description  
i    Perform case-insensitive matching  
g    Perform a global match (find all matches rather than stopping after the first match)  
m    Perform multiline matching  
  
Regular Expression Patterns  
Brackets are used to find a range of characters:  
  
Expression    Description  
[abc]    Find any of the characters between the brackets  
[0-9]    Find any of the digits between the brackets  
(x|y)    Find any of the alternatives separated with |  
Metacharacters are characters with a special meaning:  
  
Metacharacter    Description  
\d    Find a digit  
\s    Find a whitespace character  
\b    Find a match at the beginning or at the end of a word  
\uxxxx    Find the Unicode character specified by the hexadecimal number xxxx  
Quantifiers define quantities:  
  
Quantifier    Description  
n+    Matches any string that contains at least one n  
n\*    Matches any string that contains zero or more occurrences of n  
n?    Matches any string that contains zero or one occurrences of n  
  
Using the RegExp Object  
In JavaScript, the RegExp object is a regular expression object with predefined properties and methods.  
  
Using test()  
The test() method is a RegExp expression method.  
  
It searches a string for a pattern, and returns true or false, depending on the result.  
  
The following example searches a string for the character "e":  
  
Example  
  
var patt = /e/;  
patt.test("The best things in life are free!");  
Since there is an "e" in the string, the output of the code above will be:  
  
true  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Search for an "e" in the next paragraph:</p>  
  
< p id="p01">The best things in life are free!</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    text = document.getElementById("p01").innerHTML;  
    document.getElementById("demo").innerHTML = /e/.test(text);  
}  
< /script>  
  
< /body>  
< /html>  
  
You don't have to put the regular expression in a variable first. The two lines above can be shortened to one:  
  
/e/.test("The best things in life are free!")  
  
Using exec()  
The exec() method is a RegExp expression method.  
  
It searches a string for a specified pattern, and returns the found text.  
  
If no match is found, it returns null.  
  
The following example searches a string for the character "e":  
  
Example 1  
  
/e/.exec("The best things in life are free!");  
Since there is an "e" in the string, the output of the code above will be:  
  
e  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Search for an "e" in the next paragraph:</p>  
  
< p id="p01">The best things in life are free!</p>  
  
< button onclick="myFunction()">Try it</button>  
  
< p id="demo"></p>  
  
< script>  
function myFunction() {  
    text = document.getElementById("p01").innerHTML;  
    document.getElementById("demo").innerHTML = /e/.exec(text);  
}  
  
< /script>  
  
< /body>  
< /html>**

**J33**

**Hoisting is JavaScript's default behavior of moving declarations to the top.  
  
JavaScript Declarations are Hoisted  
In JavaScript, a variable can be declared after it has been used.  
  
In other words; a variable can be used before it has been declared.  
  
Example 1 gives the same result as Example 2:  
  
Example 1  
  
x = 5; // Assign 5 to x  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x;                     // Display x in the element  
  
var x; // Declare x  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
x = 5; // Assign 5 to x  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x;                     // Display x in the element  
  
var x; // Declare x  
< /script>  
  
< /body>  
< /html>  
  
  
Example 2  
  
var x; // Declare x  
x = 5; // Assign 5 to x  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x;                     // Display x in the element  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x; // Initialize x  
x = 5; // Assign 5 to x  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x;                     // Display x in the element  
< /script>  
  
< /body>  
< /html>  
  
To understand this, you have to understand the term "hoisting".  
  
Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).  
  
JavaScript Initializations are Not Hoisted  
JavaScript only hoists declarations, not initializations.  
  
Example 1 does not give the same result as Example 2:  
  
Example 1  
  
var x = 5; // Initialize x  
var y = 7; // Initialize y  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x + " " + y;           // Display x and y  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x = 5; // Initialize x  
var y = 7; // Initialize y  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x + " " + y;           // Display x and y  
  
< /script>  
  
< /body>  
< /html>  
  
  
Example 2  
  
var x = 5; // Initialize x  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x + " " + y;           // Display x and y  
  
var y = 7; // Initialize y  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x = 5;  // Initialize x  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x + " " + y;           // Display x and y  
  
var y = 7;  // Initialize y  
< /script>  
  
< /body>  
< /html>  
  
Does it make sense that y is undefined in the last example?  
  
This is because only the declaration (var y), not the initialization (=7) is hoisted to the top.  
  
Because of hoisting, y has been declared before it is used, but because initializations are not hoisted, the value of y is undefined.  
  
Example 2 is the same as writing:  
  
Example  
  
var x = 5; // Initialize x  
var y;     // Declare y  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x + " " + y;           // Display x and y  
  
y = 7;    // Assign 7 to y  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x = 5; // Initialize x  
var y;     // Declare y  
  
elem = document.getElementById("demo"); // Find an element  
elem.innerHTML = x + " " + y;           // Display x and y  
  
y = 7;     // Assign 7 to y  
  
< /script>  
  
< /body>  
< /html>  
  
  
Declare Your Variables At the Top !  
Hoisting is (to many developers) an unknown or overlooked behavior of JavaScript.  
  
If a developer doesn't understand hoisting, programs may contain bugs (errors).  
  
To avoid bugs, always declare all variables at the beginning of every scope.  
  
Since this is how JavaScript interprets the code, it is always a good rule.  
  
Note    JavaScript in strict mode does not allow variables to be used if they are not declared.  
Study "using strict"; in the next chapter.**

**J34**

**"use strict";  Defines that JavaScript code should be executed in "strict mode".  
  
The "use strict" Directive  
The "use strict" directive is new in JavaScript 1.8.5 (ECMAScript version 5).  
  
It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.  
  
The purpose of "use strict" is to indicate that the code should be executed in "strict mode".  
  
With strict mode, you cannot, for example, use undeclared variables.  
  
Note    Strict mode is supported in:  
Internet Explorer from version 10. Firefox from version 4.  
Chrome from version 13. Safari from version 5.1. Opera from version 12.  
  
Declaring Strict Mode  
Strict mode is declared by adding "use strict"; to the beginning of a JavaScript file, or a JavaScript function.  
  
Declared at the beginning of a JavaScript file, it has global scope (all code will execute in strict mode).  
  
Declared inside a function, it has local scope (only the code inside the function is in strict mode).  
  
Global declaration:  
  
"use strict";  
function testStrict(){  
    var x;  
    x = 3.14; // This does not cause an error.  
}  
x = 3.14; // This causes an error.  
Local declaration:  
  
function testStrict(){  
   "use strict";  
    x = 3.14; // This causes an error.  
}  
x = 3.14; // This does not cause an error.  
  
The "use strict"; Syntax  
The syntax, for declaring strict mode, was designed to be compatible with older versions of JavaScript.  
  
Compiling a numeric literal (4 + 5;) or a string literal ("John Doe";) in a JavaScript program has no side effects. It simply compiles to a non existing variable and dies.  
  
So "use strict;" only matters to new compilers that "understand" the meaning of it.  
  
Why Strict Mode?  
Strict mode makes it easier to write "secure" JavaScript.  
  
Strict mode changes previously accepted "bad syntax" into real errors.  
  
As an example, in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable.  
  
In normal JavaScript, a developer will not receive any error feedback assigning values to non-writable properties.  
  
In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error.  
  
Not Allowed in Strict Mode  
Using a variable (property or object) without declaring it, is not allowed:  
  
x = 3.14; // This causes an error (if x has not been declared).  
Deleting a variable, a function, or an argument, is not allowed.  
  
var testStrict = 3.14;  
delete testStrict; // This causes an error.  
Defining a property more than once, is not allowed:  
  
var testStrict = {p1:10, p2:15, p1:20}; // This causes an error.  
Duplicating a parameter name is not allowed:  
  
function testStrict(param1, param1) {}; // This causes an error.  
Octal numeric literals and escape characters are not allowed:  
  
var testStrict = 010;  // This causes an error.  
var testStrict = \010; // This causes an error.  
Writing to a read-only property is not allowed:  
  
var testObj = {};  
Object.defineProperty(testObj, "x", {value:0, writable:false});  
testObj.x = 3.14;      // This causes an error.  
Writing to a get-only property is not allowed:  
  
var testObj = {get x() {return 0} };  
testObj.x = 3.14;      // This causes an error.  
Deleting an undeletable property is not allowed:  
  
delete Object.prototype; // This causes an error.  
The string "eval" cannot be used as a variable:  
  
var eval = 3.14; // This causes an error.  
The string "arguments" cannot be used as a variable:  
  
var arguments = 3.14; // This causes an error.  
The with statement is not allowed:  
  
with (Math){x = cos(2)}; // This causes an error.  
Future reserved keywords are not allowed. These are:  
  
implements  
interface  
package  
private  
protected  
public  
static  
yield  
Other Differences  
In function calls like f(), the this value was the global object. In strict mode, it is now undefined.  
  
For security reasons, in strict mode code, eval does not create a new variable in the scope from which it was called.  
  
With strict mode, you cannot, for example, use undeclared variables.  
  
Watch Out!  
Note    The "use strict" directive is only recognized at the beginning of a script or a function.  
If you add two JavaScript files into one file, you will lose the effect of the directive in the second file.**

**J35**

**Always use the same coding conventions for all your JavaScript projects.  
  
Coding Conventions  
Coding conventions are style guidelines for programming. They typically cover:  
  
Naming and declaration rules for variables and functions.  
Rules for the use of white space, indentation, and comments.  
Programming practices and principles  
Coding conventions secure software quality:  
  
Improves code readability  
Make code maintenance easier  
Coding conventions can be documented rules for teams to follow, or just be your individual coding practice.  
  
Note    This page describes the general JavaScript code conventions used by W3Schools.  
You should also read the next chapter "Best Practices", and learn how to avoid coding pitfalls.  
Variable Names  
At W3schools we use camelCase for identifier names (variable and function). All names start with a letter.  
  
At the bottom of this page, you will find a wider discussion about naming rules.  
  
firstName = "John";  
lastName = "Doe";  
  
price = 19.90;  
discount = 0.10;  
  
fullPrice = price \* 100 / discount;  
  
Declarations on Top  
It is good coding practice to put all declarations at the top of each script or function.  
  
This gives better, cleaner code, and reduces the possibility of accidental re-declarations.  
  
var firstName, lastName;  
var price, discount, fullPrice;  
  
firstName = "John";  
lastName = "Doe";  
  
price = 19.90;  
discount = 0.10;  
  
fullPrice = price \* 100 / discount;  
This also goes for variables in loops:  
  
var i;  
for (i = 0; i < 5; i++)  
Note    Since JavaScript moves the declarations to the top anyway (JavaScript hoisting), it is always a good rule.  
Spaces Around Operators  
Always put spaces around operators, and after commas:  
  
x = 5 + 6;        // Good  
x=5+6             // Bad  
  
[40, 100, 1, 5]   // Good  
[40,100,1,5]      // Bad  
  
Code Indentation  
Always use 4 spaces for indentation of code blocs:  
  
function toCelsius(fahrenheit) {  
    return (5/9) \* (fahrenheit-32);  
}  
  
  
for (i = 1; i < 50; i++) {  
    sum += i;  
}  
Note    Do not use tabs (tabulators) for indentation. Text editors interpret tabs differently.  
Line Length < 80  
For readability, avoid lines longer than 80 characters.  
  
If a JavaScript statement does not fit on one line, the best place to break it, is after an operator or a comma.  
  
Example  
  
document.getElementById("demo").innerHTML =  
    "Hello Dolly.";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h1>My Web Page</h1>  
  
< p>  
The best place to break a code line is after an operator or a comma.  
< /p>  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML =  
    "Hello Dolly.";  
< /script>  
  
< /body>  
< /html>  
  
  
Performance  
Coding conventions are not used by computers. Most rules have little impact on the execution of programs.  
  
Indentation and extra spaces are not significant in small scripts.  
  
For code in development, readability should be preferred. Larger production scripts should be minifyed.  
  
Naming Conventions  
Always use the same naming convention for all your code. For example:  
  
Variable and function names written as camelCase  
Global variable written in UPPERCASE  
Constants (like PI) written in UPPERCASE  
Should you use hyp-hens, camelCase, or under\_scores in variable names?  
  
This is a question programmers often discuss. The answer depends on who you ask:  
  
Hyphens in HTML and CSS:  
  
HTML5 attributes can start with data- (data-quantity, data-price).  
  
CSS uses hyphens in property-names (font-size).  
  
Note    Hyphens can be mistaken as subtraction attempts. Hyphens are not allowed in JavaScript names.  
Underscores:  
  
Many programmers prefer to use underscores (date\_of\_birth), especially in SQL databases.  
  
Underscores are often used in PHP documentation.  
  
CamelCase:  
  
CamelCase is often preferred by C programmers.  
  
camelCase:  
  
camelCase is used by JavaScript itself, by jQuery, and other JavaScript libraries.  
  
Note    Don't start names with a $ sign. It will put you in conflict with many JavaScript library names.**

**J36**

**Avoid global variables,  avoid new,  avoid  ==,  avoid eval()  
  
Avoid Global Variables  
Avoid using global variables.  
  
This includes all data types, objects, and functions.  
  
Global variables and functions can be overwritten by other scripts.  
  
Use local variables instead, and learn how to use closures.  
  
Always Declare Local Variables  
All variables used in a function should be declared as local variables.  
  
Local variables must be declared with the var keyword, otherwise they will become global variables.  
  
Note    Strict mode does not allow undeclared variables.  
Never Declare Numbers, Strings, or Booleans as Objects  
Always treat numbers, strings, or booleans as primitive values. Not as objects.  
  
Declaring numbers, strings, or booleans as objects, slows down execution speed, and produces nasty side effects:  
  
Example  
  
var x = "John";  
var y = new String("John");  
(x === y) // is false because x is a string and y is an object.  
  
< !DOCTYPE html>  
< html>  
< body>  
< p id="demo"></p>  
  
< script>  
var x = "John";              // x is a string  
var y = new String("John");  // y is an object  
document.getElementById("demo").innerHTML = x===y;  
< /script>  
  
< /body>  
< /html>  
  
  
Don't Use new Object()  
Use {} instead of new Object()  
Use "" instead of new String()  
Use 0 instead of new Number()  
Use false instead of new Boolean()  
Use [] instead of new Array()  
Use /(:)/ instead of new RegExp()  
Use function (){} instead of new function()  
Example  
  
var x1 = {};           // new object  
var x2 = "";           // new primitive string  
var x3 = 0;            // new primitive number  
var x4 = false;        // new primitive boolean  
var x5 = [];           // new array object  
var    x6 = /()/          // new regexp object  
var x7 = function(){}; // new function object  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x1 = {};  
var x2 = "";  
var x3 = 0;  
var x4 = false;  
var x5 = [];  
var x6 = /()/;  
var x7 = function(){};  
  
document.getElementById("demo").innerHTML =  
"x1: " + typeof x1 + "<br>" +  
"x2: " + typeof x2 + "<br>" +  
"x3: " + typeof x3 + "<br>" +  
"x4: " + typeof x4 + "<br>" +  
"x5: " + typeof x5 + "<br>" +  
"x6: " + typeof x6 + "<br>" +  
"x7: " + typeof x7 + "<br>";  
< /script>  
  
< /body>  
< /html>  
  
  
Beware of Automatic Type Conversions  
Beware that numbers can accidentally be converted to strings or NaN (Not a Number).  
  
JavaScript is loosely typed. A variable can contain different data types, and a variable can change its data type:  
  
Example  
  
var x = "Hello";     // typeof x is a string  
x = 5;               // changes typeof x to a number  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x = "Hello";  
var x = 5;  
document.getElementById("demo").innerHTML = typeof x;  
< /script>  
  
< /body>  
< /html>  
  
  
When doing mathematical operations, JavaScript can convert numbers to strings:  
  
Example  
  
var x = 5 + 7;       // x.valueOf() is 12,  typeof x is a number  
var x = 5 + "7";     // x.valueOf() is 57,  typeof x is a string  
var x = "5" + 7;     // x.valueOf() is 57,  typeof x is a string  
var x = 5 - 7;       // x.valueOf() is -2,  typeof x is a number  
var x = 5 - "7";     // x.valueOf() is -2,  typeof x is a number  
var x = "5" - 7;     // x.valueOf() is -2,  typeof x is a number  
var x = 5 - "x";     // x.valueOf() is NaN, typeof x is a number  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var x = "Hello";  
var x = 5;  
document.getElementById("demo").innerHTML = typeof x;  
< /script>  
  
< /body>  
< /html>  
  
  
Subtracting a string from a string, does not generate an error but returns NaN (Not a Number):  
  
Example  
  
"Hello" - "Dolly"    // returns NaN  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo">My first paragraph.</p>  
  
< script>  
document.getElementById("demo").innerHTML = "Hello" - "Dolly";  
< /script>  
  
< /body>  
< /html>  
  
  
Use === Comparison  
The == comparison operator always converts (to matching types) before comparison.  
  
The === operator forces comparison of values and type:  
  
Example  
  
0 == "";        // true  
1 == "1";       // true  
1 == true;      // true  
  
0 === "";       // false  
1 === "1";      // false  
1 === true;     // false  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Remove the comment (at the beginning of each line) to test each case:</p>  
  
< p id="demo"></p>  
  
< script>  
var x;  
//x = (0 == "");     // true  
//x = (1 == "1");    // true  
//x = (1 == true);   // true  
//x = (0 === "");    // false  
//x = (1 === "1");   // false  
//x = (1 === true);  // false  
document.getElementById("demo").innerHTML = x;  
< /script>  
  
< /body>  
< /html>  
  
  
  
Never End a Definition with a Comma  
Bad Examples  
  
points = [40, 100, 1, 5, 25, 10, ];  
  
person = {firstName:"John", lastName:"Doe", age:46, }  
Some JSON and JavaScript engines will fail, or behave unexpectedly.  
  
Use Parameter Defaults  
If a function is called with a missing argument, the value of the missing argument is set to undefined.  
  
Undefined values can break your code. It is a good habit to assign default values to arguments.  
  
Example  
  
function myFunction(x, y) {  
    if (y === undefined) {  
        y = 0;  
    }  
}  
Or, even simpler:  
  
function myFunction(x, y) {  
    y = y || 0;  
}  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p>Setting a default value to a function parameter.</p>  
< p id="demo"></p>  
  
< script>  
function myFunction(x, y) {  
    y = y || 0;  
    return x \* y;  
}  
document.getElementById("demo").innerHTML = myFunction(4);  
< /script>  
  
< /body>  
< /html>  
  
  
  
Read more about function parameters and arguments at Function Parameters  
  
Avoid Using eval()  
The eval() function is used to run text as code. In almost all cases, it should not be necessary to use it.  
  
Because it allows arbitrary code to be run, it also represents a security problem.**

**J37**

**How to speed up your JavaScript code.  
  
Reduce Activity in Loops  
Loops are often used in programming.  
  
Every statement inside a loop will be executed for each iteration of the loop.  
  
Search for statements or assignments that can be placed outside the loop.  
  
Reduce DOM Access  
Accessing the HTML DOM is very slow, compared to other JavaScript statements.  
  
If you expect to access a DOM element several times, access it once, and use it as a local variable:  
  
Example  
  
obj = document.getElementByID("demo");  
obj.innerHTML = "Hello";  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< p id="demo"></p>  
  
< script>  
var obj;  
obj = document.getElementById("demo");  
obj.innerHTML = " Hello";  
< /script>  
  
< /body>  
< /html>  
  
  
Reduce DOM Size  
Keep the number of elements in the HTML DOM small.  
  
This will always improve page loading, and speed up rendering (page display), especially on smaller devices.  
  
Every attempt to search the DOM (like getElementsByTagName) is will benefit from a smaller DOM.  
  
Avoid Unnecessary Variables  
Don't create new variables if you don't plan to save values.  
  
Often you can replace code like this:  
  
var fullName = firstName + " " + lastName;  
document.getElementById("demo").innerHTML = fullName;  
With this:  
  
document.getElementById("demo").innerHTML = firstName + " " + lastName  
  
Delay JavaScript Loading  
Putting your scripts at the bottom of the page body, lets the browser load the page first.  
  
While a script is downloading, the browser will not start any other downloads. In addition all parsing and rendering activity might be blocked.  
  
Note    The HTTP specification defines that browsers should not download more than two components in parallel.  
An alternative is to use defer="true" in the script tag. The defer attribute specifies that the script should be executed before the page has finished parsing, but it only works for external scripts.  
  
If possible, you can add your script to the page by code, after the page has loaded:  
  
Example  
  
< script>  
window.onload = downScripts;  
  
function downScripts() {  
    var element = document.createElement("script");  
    element.src = "myScript.js";  
    document.body.appendChild(element);  
}  
< /script>  
  
Avoid Using with  
Avoid using the with keyword. It has a negative effect on speed. It also clutters up JavaScript scopes.  
  
The with keyword is not allowed in strict mode.**

**J38**

**In JavaScript, some identifiers are reserved words and cannot be used as variables or function names.  
  
JavaScript Standards  
All modern browsers fully support ECMAScript 3 (ES3, the third edition of JavaScript from 1999).  
  
ECMAScript 4 (ES4) was never adopted.  
  
ECMAScript 5 (ES5, released in 2009) is the latest official version of JavaScript.  
  
Time passes, and we are now beginning to see complete support for ES5 in all modern browsers.  
  
JavaScript Reserved Words  
In JavaScript you cannot use these reserved words as variables, labels, or function names:  
  
abstract    arguments    boolean    break    byte  
case    catch    char    class\*    const  
continue    debugger    default    delete    do  
double    else    enum\*    eval    export\*  
extends\*    false    final    finally    float  
for    function    goto    if    implements  
import\*    in    instanceof    int    interface  
let    long    native    new    null  
package    private    protected    public    return  
short    static    super\*    switch    synchronized  
this    throw    throws    transient    true  
try    typeof    var    void    volatile  
while    with    yield  
Words marked with\* are new in ECMAScript5  
  
JavaScript Objects, Properties, and Methods  
You should also avoid using the name of JavaScript built-in objects, properties, and methods:  
  
Array    Date    eval    function    hasOwnProperty  
Infinity    isFinite    isNaN    isPrototypeOf    length  
Math    NaN    name    Number    Object  
prototype    String    toString    undefined    valueOf  
  
Java Reserved Words  
JavaScript is often used together with Java. You should avoid using some Java objects and properties as JavaScript identifiers:  
  
getClass    java    JavaArray    javaClass    JavaObject    JavaPackage  
  
Windows Reserved Words  
JavaScript can be used outside HTML. It can be used as the programming language in many other applications.  
  
In HTML you must (for portability you should) avoid using the name of HTML and Windows objects and properties:  
  
alert    all    anchor    anchors    area  
assign    blur    button    checkbox    clearInterval  
clearTimeout    clientInformation    close    closed    confirm  
constructor    crypto    decodeURI    decodeURIComponent    defaultStatus  
document    element    elements    embed    embeds  
encodeURI    encodeURIComponent    escape    event    fileUpload  
focus    form    forms    frame    innerHeight  
innerWidth    layer    layers    link    location  
mimeTypes    navigate    navigator    frames    frameRate  
hidden    history    image    images    offscreenBuffering  
open    opener    option    outerHeight    outerWidth  
packages    pageXOffset    pageYOffset    parent    parseFloat  
parseInt    password    pkcs11    plugin    prompt  
propertyIsEnum    radio    reset    screenX    screenY  
scroll    secure    select    self    setInterval  
setTimeout    status    submit    taint    text  
textarea    top    unescape    untaint    window  
  
HTML Event Handlers  
In addition you should avoid using the name of all HTML event handlers.  
  
Examples:  
  
onblur    onclick    onerror    onfocus  
onkeydown    onkeypress    onkeyup    onmouseover  
onload    onmouseup    onmousedown    onsubmit  
  
Nonstandard JavaScript  
In addition to reserved words, there are also some nonstandard keywords used in some JavaScript implementations.  
  
One example is the const keyword used to define variables. Some JavaScript engines will treat const as a synonym to var. Other engines will treat const as a definition for read-only variables.  
  
Const is an extension to JavaScript. It is supported by the JavaScript engine used in Firefox and Chrome. But it is not a part of the JavaScript standards ES3 or ES5. Do not use it.**

**J39**

**JSON is a format for storing and transporting data.  
  
JSON is often used when data is sent from a server to a web page.  
  
What is JSON?  
JSON stands for JavaScript Object Notation  
JSON is lightweight data interchange format  
JSON is language independent \*  
JSON is "self-describing" and easy to understand  
Note    \* JSON uses JavaScript syntax, but the JSON format is text only.  
Text can be read and used as a data format by any programming language.  
JSON Example  
This JSON syntax defines an employees object: an array of 3 employee records (objects):  
  
JSON Example  
  
{"employees":[  
    {"firstName":"John", "lastName":"Doe"},  
    {"firstName":"Anna",    "lastName":"Smith"},  
    {"firstName":"Peter", "lastName":"Jones"}  
]}  
  
The JSON Format Evaluates to JavaScript Objects  
The JSON format is syntactically identical to the code for creating JavaScript objects.  
  
Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.  
  
JSON Syntax Rules  
Data is in name/value pairs  
Data is separated by commas  
Curly braces hold objects  
Square brackets hold arrays  
JSON Data - A Name and a Value  
JSON data is written as name/value pairs, Just like JavaScript object properties.  
  
A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:  
  
"firstName":"John"  
  
JSON Objects  
JSON objects are written inside curly braces.  
  
Just like in JavaScript, objects can contain multiple name/values pairs:  
  
{"firstName":"John", "lastName":"Doe"}  
  
JSON Arrays  
JSON arrays are written inside square brackets.  
  
Just like in JavaScript, an array can contain objects:  
  
"employees":[  
    {"firstName":"John", "lastName":"Doe"},  
    {"firstName":"Anna", "lastName":"Smith"},  
    {"firstName":"Peter", "lastName":"Jones"}  
]  
In the example above, the object "employees" is an array. It contains three objects.  
  
Each object is a record of a person (with a first name and a last name).  
  
Converting a JSON Text to a JavaScript Object  
A common use of JSON is to read data from a web server, and display the data in a web page.  
  
For simplicity, this can be demonstrated using a string as input (or read more in our JSON tutorial):  
  
First, create a JavaScript string containing JSON syntax:  
  
var text = '{ "employees" : [' +  
'{ "firstName":"John" , "lastName":"Doe" },' +  
'{ "firstName":"Anna" , "lastName":"Smith" },' +  
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';  
Then, use the JavaScript built-in function JSON.parse() to convert the string into a JavaScript object:  
  
var obj = JSON.parse(text);  
Finally, use the new JavaScript object in your page:  
  
Example  
  
< p id="demo"></p>  
  
< script>  
document.getElementById("demo").innerHTML =  
obj.employees[1].firstName + " " + obj.employees[1].lastName;  
< /script>  
  
< !DOCTYPE html>  
< html>  
< body>  
  
< h2>Create Object from JSON String</h2>  
  
< p id="demo"></p>  
  
< script>  
var text = '{"employees":[' +  
'{"firstName":"John","lastName":"Doe" },' +  
'{"firstName":"Anna","lastName":"Smith" },' +  
'{"firstName":"Peter","lastName":"Jones" }]}';  
  
obj = JSON.parse(text);  
document.getElementById("demo").innerHTML =  
obj.employees[1].firstName + " " + obj.employees[1].lastName;  
< /script>  
  
< /body>  
< /html>**