How to put a JavaScript into an HTML document

```
<html>
<head>
</head>
<body>
<script language="JavaScript">
document.write("Hello World!")
</script>
</body>
</html>
```

And it produces this output:

```
Hello World!
```

To insert a script in an HTML document, use the <script> tag. Use the language attribute to define the scripting language.

```
<script language="JavaScript">
```

Then comes the JavaScript: In JavaScript the command for writing some text on a page is **document.write**:

```
document.write("Hello World!")
```

The script ends:

```
</script>
```

How to handle older browsers

Older browsers that do not support scripts will display the script as page content. To prevent them from doing this, you can use the HTML comment tag:

```
<script language="JavaScript">
<!--
    some statements
//-->
</script>
```

The two forward slashes on front of the end of comment line (//-->) are a JavaScript comment symbol, and prevent the JavaScript from trying to compile the line.

Note that you cannot put // in front of the first comment line (like //<!--), because older browser will display it. Funny? Yes ! But that's the way it is.

Where to put the JavaScript

Scripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user trigger an event.

Scripts in the head section: Scripts to be executed when they are called, or when an event is triggered, goes in the head section. When you place a script in the head section, you will ensure that the script is loaded before anyone uses it.

```
<html>
<head>
<script language="JavaScript">
     some statements
</script>
</head>
```

Scripts in the body section: Scripts to be executed when the page loads, goes in the body section. When you place a script in the body section it generates the content of the page.

```
<html>
<head>
</head>
<body>
<script language="JavaScript">
     some statements
</script>
</body>
```

Scripts in both the body and the head section: You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

```
<html>
<head>
<script language="JavaScript">
     some statements
</script>
</head>
<body>
<script language="JavaScript">
     some statements
</script>
</body>
```

How to use external script

Sometimes you might want to run the same script on several pages, without writing the script on each and every page. To simplify this you can write the script in a external file, and save it with a .js file extension. Like this:

```
document.write("This script is external")
```

Save the external file as xxx.js
-The filename can not contain more than 8 letters
-The external script can not contain the <script> tag

Now you can call this script, using the "src" attribute, from any of your pages:

```
<html>
<head>
</head>
```

```
<body>
<script src="xxx.js">
</script>
</body>
</html>
```

Remember to place the script exactly where you normally would write the script.

Variables

A variable is a "container" for information you want to store. A variable's value can change during the script. You can refer to a variable by name to see its value or to change its value.

Rules for Variable names:

- Variable names are case sensitive
- They must begin with a letter or the underscore character

Declaring Variables

You can create a variable with the var statement:

```
var strname = some value
```

You can also create a variable without the var statement:

```
strname = some value
```

Assigning Values to Variables

You assign a value to a variable like this:

```
var strname = "Hege"
```

Or like this:

```
strname = "Hege"
```

The variable name is on the left side of the expression and the value you want to assign to the variable is on the right. Now the variable "strname" has the value "Hege".

Assignment Operators

Operator	Example	Result
=	i = 5	i equals 5
+=	i += 5	i equals i + 5
-=	i -= 5	i equals i - 5
*=	i *= 5	i equals i * 5

/=	i /= 5	i equals i / 5
%=	i %= 5	i equals i %5
++	i++	i equals i+1
		i equals i-1

Lifetime of Variables

When you declare a variable within a function, only code within that function can access or change the value of that variable. When the function exits, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.

If you declare a variable outside a function, all the functions in your script will recognize it. These variables exists from the time they are declared until the time the script is finished running.

The Boolean object

The Boolean object is one of JavaScript's built-in objects. It can return two values: 0 for false and any other integer value for true. It is used to return a result of a conditional test. For example if you want some code to be executed if a condition is false/true.

Methods	**Explanation**	**NN**	**IE**	**ECMA**
toString()	Returns a string Boolean value. (true or false)	3.0	3.0	1.0
valueOf()		4.0	J3	1.0

Functions

A function contains some code that will be executed by an event or a call to that function. A function is a set of statements. You can reuse functions within the same script, or in other documents. You define functions at the beginning of a file (in the head section), and call them later in the document. It is now time to take a lesson about the alert-box:

This is JavaScript's method to alert the user.

```
alert("here goes the message")
```

How to define a function

To create a function you define its name, any values ("arguments"), and some statements:

```
function myfunction(argument1,argument2,etc)
{
some statements
}
```

A function with no arguments must include the brackets:

```
function myfunction()
{
```

```
some statements
}
```

Arguments are variables that will be used in the function. The variable values will be the values passed on by the function call.

By placing functions in the head section of the document, you make sure that all the code in the function has been loaded before the function is called.

Some functions returns a value to the calling expression

```
function result(a,b)
{
c=a+b
return c
}
```

How to call a function

A function is not executed before it is called.

You can call a function containing arguments:

```
myfunction(argument1,argument2,etc)
```

or without arguments:

```
myfunction()
```

The return statement

Functions that will return a result, must use the return statement, this statement specifies the value which will be returned to where the function was called from. Say you have a function that returns the sum of two numbers:

```
function total(a,b)
{
result=a+b
return result
}
```

When you call this function you must send two arguments with it:

```
sum=total(2,3)
```

Conditionals

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 two conditional statements:

- **if...else statement** - use this statement if you want to select one of two sets of lines to execute
- **switch statement** - use this statement if you want to select one of many sets of lines to execute

---

If Condition

You should use the this statement if you want to execute some code if a condition is true, or if you want to select one of two blocks of code to execute.

If you want to execute only one statement when a condition is true, use this syntax for the if...else statement, like this:

```
if (condition)
{
code to be executed if condition is true
}
```

Notice that there is no ..else.. in this syntax. You just tell the code to perform **one action** if the condition is true.

If you want to execute some statements if a condition is true and execute others if a condition is false, use this syntax for the if....else statement, like this:

```
if (condition)
{
code to be executed if condition is true
}
else
{
code to be executed if condition is false
}
```

Equality Operators

Operator	Meaning
==	is equal to
!=	is not equal to
>	is greater than
>=	is greater than or equal to
<	is less than
<=	is less than or equal to

Switch Condition

You should use the this statement if you want to select one of many blocks of code to execute.

```
switch (expression)
```

```
{
case label1:
        code to be executed if expression = label1
        break
case label2:
        code to be executed if expression = label2
        break
default:
        code to be executed
        if expression is different
        from both label1 and label2
}
```

Looping

Very often when you write code, you want allow the same block of code to run a number of times. You can use looping statements in your code to do this.

In JavaScript we have the following looping statements:

- **while** - loops through a block of code while a condition is true
- **do...while** - loops through a block of code once, and then it repeats the loop while a condition is true
- **for** - run statements a specified number of times

---

while

The while statement will execute a block of code while a condition is true..

```
while (condition)
{
    code to be executed
}
```

do...while

The do...while statement will execute a block of code once, and then it will repeat the loop while a condition is true

```
do
{
    code to be executed
}
while (condition)
```

for

The for statement will execute a block of code a specified number of times

```
for (initialization; condition; increment)
{
    code to be executed
}
```

JavaScript is Case Sensitive

A function named "myfunction" is not the same as "myFunction". Therefore watch your capitalization when you create or call variables, objects and functions.

Symbols

Open symbols, like ({ [" ', must have a matching closing symbols, like ' "] }).

White Space

JavaScript ignores extra spaces. You can add white space to your script to make it more readable. These two lines means exactly the same:

```
name="Hege"
name = "Hege"
```

Break up a Code line

You can break up a code line with the backslash. The example below will be displayed properly:

```
document.write \
("Hello World!")
```

You can insert special characters (like " ' ; &) with the backslash:

```
document.write ("You \& I sing \"Happy Birthday\".")
```

The example above will produce this output:

```
You & I sing "Happy Birthday".
```

Comments

You can add a comment to your JavaScript code starting the comment with "//":

```
sum=a + b  //calculating the sum
```

You can also add a comment to the JavaScript code, starting the comment with "/*" and ending it with "*/"

```
sum=a + b  /*calculating the sum*/
```

Using "/*" and "*/" is the only way to create a multi-line comment:

```
/* This is a comment
block. It contains
several lines*/
```

Examples

The length() Method
Use the length property to find out how many character there is in the string.

The indexOf() Method
Test if a string contains a specified character. Returns an integer if it does and -1 if it do not. Use this method in a form validation.

The match() Method
Works similar to the indexOf method, only this method returns the characters you specified, "null" if the string do not contain the specified characters.

The substr() Method
The substr method returns specified parts of the string. If you specify (14,7) the return will be the 14th character and the next 7. Note that the first character is 0, the second is 1 etc.

The toLowerCase() and toUpperCase() Methods
How to return a string in lower or upper case.

The most common methods

Methods	**Explanation**	**NN**	**IE**	**ECMA**
length	Returns the length of the string	2.0	3.0	1.0
indexOf()	Returns the index of the first time the specified character occurs, or -1 if it never occurs, so with that index you can determine if the string contains the specified character.	2.0	3.0	
lastIndexOf()	Same as indexOf, only it starts from the right and moves left.	2.0	4.0	
match()	Behaves similar to indexOf and lastIndexOf, but the match method returns the specified characters, or "null", instead of a numeric value.	4.0	4.0	
substr()	Returns the characters you specified: (14,7) returns 7 characters, from the 14th character.	4.0	4.0	
substring()	Returns the characters you specified: (14,7) returns all characters between the 7th and the 14th.	2.0	3.0	1.0
toLowerCase()	Returns the string in lower case	2.0	3.0	1.0
toUpperCase()	Returns the string in upper case	2.0	3.0	1.0

The Array object

Some times you want to assign more than one value to a single variable. Then you can create a variable that can contain a series of values. This is called an array variable. The declaration of an array variable:

```
names = new Array(3)
```

The expecting number of elements goes inside the parentheses, in this case 3. You assign data to each of the elements of the array like this:

```
names[0] = "Tove"
names[1] = "Jani"
names[2] = "Ståle"
```

Similarly, the data can be retrieved from any element using an index into the particular array element you want. Like this:

```
mother = names[0]
```

The Date object

The Date object is one of JavaScripts built-in objects. It is used to return a date. To do this you must declare a variable:

```
mydate=new Date()
```

You can write a date inside the parentheses, if not, the "mydate" variable contains today's date, see the Date() example below. Now you can access all the methods in the Date object from this variable. Example:

```
mydate.getDay()
```

The most common methods

Methods	Explanation	NN	IE	ECMA
Date()	Returns a new Date object	2.0	3.0	1.0
getDate()	Returns the date from a Date object. (1-31)	2.0	3.0	1.0
getDay()	Returns the weekday. (0-6)	2.0	3.0	1.0
getMonth()	Returns the month. (0-11)	2.0	3.0	1.0
getFullYear()	Returns the year. (2000)	4.0	4.0	1.0
getHours()	Returns the hour as a value between 0 and 23	2.0	3.0	1.0
getMinutes()	Returns the minute. (0-59)	2.0	3.0	1.0
getSeconds()	Returns the second. (0-59)	2.0	3.0	1.0

The Math object

A built-in object that let you perform mathematic functions. This object is a bit different from the other objects in JavaScript. For example, the Math object is not created with the "new" operator, if you do so, an error will occur. This is how you create the object:

```
Math.random()
```

The most common methods

Methods	Explanation	NN	IE	ECMA
max()	Returns the number with the highest value of two numbers	2.0	3.0	1.0
min()	Returns the number with the lowest value of two numbers	2.0	3.0	1.0
random()	Returns a random number	2.0	3.0	1.0
round()	Returns a number rounded to the nearest whole number	2.0	3.0	1.0

Browser:
<html>
<head>

```
<script language="JavaScript">
document.write("You are browsing this site with: "+ navigator.appName)
</script>

</head>
<body>
</body>
</html>
```

Message:

```
<html>
<head>
<script language="JavaScript">
function browserversion()
{
txt="Your Browser is unknown"
browser=navigator.appVersion
if (browser.indexOf("2.")>-1)
{
txt="Your Browser is from the stone-age"
}
if (browser.indexOf("3.")>-1)
{
txt="You should update your Browser."
}
if (browser.indexOf("4.")>-1)
{
txt="Your Browser is good enough"
}
document.forms[0].message.value=txt
}
</script>
</head>

<body onload="browserversion()">

<form>
<input type="text" name="message" size="50">
</form>

</body>
</html>
```

Verschiedene Browser, verschiedene Seiten:

```
<html>
<head>
<script language="JavaScript">
function redirectme()
{
bname=navigator.appName
if (bname.indexOf("Netscape")!=-1)
        {
        window.location="tryjs_netscape.htm"
        return
        }
if (bname.indexOf("Microsoft")!=-1)
        {
        window.location="tryjs_microsoft.htm"
        return
        }
window.location="tryjs_other.htm"
}
</script>
</head>
<body>
<form>
<input type="button" onclick="redirectme()" value="Redirect">
</form>
</body>
</html>
```

E-Mail schicken:

```
<html>
<head>
<script language="JavaScript">
function validate()
{
at=document.forms[0].email.value.indexOf("@")

if (at<0)
{
alert("Not a valid e-mail")
document.forms[0].action="tryjs_email.htm"
}
}
</script>
</head>
<body>
<form onsubmit="validate()" action="tryjs_rightpage.htm">
<input type="text" name="email" size="30">
<input type="submit" value="Validate">
</form>
</body>
</html>
```

Change Url:

```
<html>

<frameset cols="70%,*" frameborder="1">
        <frame name="leftframe" src="tryjs_threeframes_sourcepage.htm">
        <frameset rows="50%,*" frameborder="1">
                <frame name="upperframe" src="../html/tryhtml_frame_a.htm">
                <frame name="lowerframe" src="../html/tryhtml_frame_b.htm">
        </frameset>
</frameset>

</html>
```

```
<html>
<head>
<script language="JavaScript">
function changeurl()
{
parent.upperframe.location.href="html_frame_c.htm"
parent.lowerframe.location.href="html_frame_d.htm"
}
</script>
</head>
<body>
<form>
<input type="button" value="Change url"
onclick="changeurl()">
</form>
</body>
</html>
```

Alert:

```
<html>
<body>

<script language="JavaScript">
alert("Hello World!")
</script>

</body>
</html>
```

Confirm Box:

```
<html>
<body>

<script language="JavaScript">
var name = confirm("Press a button")
if (name == true)
{
document.write("You pressed OK")
}
else
{
document.write("You pressed Cancel")
}
</script>

</body>
```

</html>

The Boolean object

Methods	Explanation	NN	IE	ECMA
toString()	Returns a string Boolean value. (true or false)	3.0	3.0	1.0
valueOff()	Returns the value of the specified object	4.0	4.0	1.0

The String object

Methods	Explanation	NN	IE	ECMA
length	Returns the length of the string	2.0	3.0	1.0
anchor()	Returns the string as an anchor: <a>*string*</a>	2.0	3.0	
big()	Returns the string, formatted to big: <big>*string*</big>	2.0	3.0	
blink()	Returns the string blinking: <blink>*string*</blink>	2.0		
bold()	Returns the string bold: <b>*string*</b>	2.0	3.0	
charAt()	Returns the character at the specified index	2.0	3.0	1.0
charCodeAt()	Returns the Unicode of the specified indexed character	4.0	4.0	1.0
concat()	Returns two concatenated strings	4.0	4.0	
fixed()	Returns the string as teletype text: <tt>*string*</tt>	2.0	3.0	
fontColor()	Returns the string in the specified color: <font color="*red*">*string*</font>	2.0	3.0	
fontSize()	Returns the string in the specified size: <font size="*5*">*string*</font>	2.0	3.0	
fromCharCode()	The charCodeAt method vice versa. Returns the string value of the specified Unicode.	4.0	4.0	
indexOf()	Returns the index of the first time the specified character occurs, or -1 if it never occurs, so with that index you can determine if the string contains the specified character.	2.0	3.0	
italics()	Returns the string in italic: <i>*string*</i>	2.0	3.0	
lastIndexOf()	Same as indexOf, only it starts from the right and moves left.	2.0	3.0	
link()	Returns the string as a hyperlink: <a href="*url*">*string*</a>	2.0	3.0	
match()	Behaves similar to indexOf and lastIndexOf, but the match method returns the specified characters, or "null", instead of a numeric value.	4.0	4.0	
replace()	Replaces the specified characters with new specified characters.	4.0	4.0	
search()	Returns an integer value if the string contains the specified characters, if not it returns -1.	4.0	4.0	
slice()	Returns a string containing the specified character index.	4.0	4.0	
small()	Returns the string formatted to small: <small>*string*</small>	2.0	3.0	
split()	Replaces the specified characters with a comma.	4.0	4.0	1.0
strike()	Returns the string strikethrough: <strike>*string*</strike>	2.0	3.0	
sub()	Returns the string formatted to subscript: <sub>*string*</sub>	2.0	3.0	
substr()	Returns the characters you specified. (14,7) returns 7 characters, from the 14th character.	4.0	4.0	
substring()	Returns the characters you specified. (14,7) returns all	2.0	3.0	1.0

	characters between the 7th and the 14th.			
sup()	Returns the string formatted to superscript: <sup>*string*</sup>	2.0	3.0	
toLowerCase()	Returns the string in lower case	2.0	3.0	1.0
toUpperCase()	Returns the string in upper case	2.0	3.0	1.0

The Array object

Methods	**Explanation**	**NN**	**IE**	**ECMA**
length	Returns number of elements in the array	3.0	4.0	1.0
concat()	Returns an array concatenated of two arrays	4.0	4.0	1.0
join()	Returns a string of all the elements of an array concatenated together	3.0	4.0	1.0
reverse()	Returns the array reversed	3.0	4.0	1.0
slice()	Returns a specified part of the array	4.0	4.0	
sort()	Returns a sorted array	3.0	4.0	1.0

The Date Object

Methods	**Explanation**	**NN**	**IE**	**ECMA**
Date()	Returns a new Date object	2.0	3.0	1.0
getDate()	Returns the date from a Date object. (1-31)	2.0	3.0	1.0
getDay()	Returns the weekday. (0-6)	2.0	3.0	1.0
getMonth()	Returns the month. (0-11)	2.0	3.0	1.0
getFullYear()	Returns the year. (2000)	4.0	4.0	1.0
getYear()	Returns the year as a 4 digit value (or the year as a 2 digit value if the date is before January 1, 2000). Use getFullYear instead !!	2.0	3.0	1.0
getHours()	Returns the hour as a value between 0 and 23	2.0	3.0	1.0
getMinutes()	Returns the minute. (0-59)	2.0	3.0	1.0
getSeconds()	Returns the second. (0-59)	2.0	3.0	1.0
getMilliseconds()	Returns the millisecond. (0-999)	4.0	4.0	1.0
getTime()	Returns the number of milliseconds since 1/1-1970	2.0	3.0	1.0
getTimezoneOffset()	Returns the time difference between the user's computer and GMT	2.0	3.0	1.0
getUTCDate()	Returns the date as set by the World Time Standard, UTC = Universal Coordinated Time. To return the local time use the getDate method	4.0	4.0	1.0
getUTCDay()	Returns the UTC day	4.0	4.0	1.0
getUTCMonth()	Returns the UTC month	4.0	4.0	1.0
getUTCFullYear()	Returns the UTC 4 digit year	4.0	4.0	1.0
getUTCHourc()	Returns the UTC hour	4.0	4.0	1.0
getUTCMinutes()	Returns the UTC minutes	4.0	4.0	1.0
getUTCSeconds()	Returns the UTC seconds	4.0	4.0	1.0
getUTCMilliseconds()	Returns the UTC milliseconds	4.0	4.0	1.0

parse()	Returns a string date value into how many milliseconds since 1/1-1970.	2.0	3.0	1.0
setDate()	Sets a new date into the Date object	2.0	3.0	1.0
setFullYear()	Sets a new year into the Date object	4.0	4.0	1.0
setHours()	Sets a new hour into the Date object. (0-23)	2.0	3.0	1.0
setMilliseconds()	Sets new milliseconds into the Date object. (0-999)	4.0	4.0	1.0
setMinutes()	Sets mew minutes hour into the Date object. (0-59)	2.0	3.0	1.0
setMonth()	Sets a new month into the Date object. (0-11)	2.0	3.0	1.0
setSeconds()	Sets new seconds into the Date object. (0-59)	2.0	3.0	1.0
setTime()	Sets milliseconds after 1/1-1970	2.0	3.0	1.0
setYear()	Sets a new year into the Date object	2.0	3.0	1.0
setUTCDate()	Sets The UTC date	4.0	4.0	1.0
setUTCDay()	Sets The UTC date	4.0	4.0	1.0
setUTCMonth()	Sets The UTC date	4.0	4.0	1.0
setUTCFullYear()	Sets The UTC date	4.0	4.0	1.0
setUTCHourc()	Sets The UTC date	4.0	4.0	1.0
setUTCMinutes()	Sets The UTC date	4.0	4.0	1.0
setUTCSeconds()	Sets The UTC date	4.0	4.0	1.0
setUTCMilliseconds()	Sets The UTC date	4.0	4.0	1.0
toGMTString()	Returns a string date value.	2.0	3.0	1.0
toLocaleString()	Returns a string date value.	2.0	3.0	1.0
toString()	Returns a string date value.	2.0	4.0	1.0

The Math object

Properties	Explanation	NN	IE	ECMA
E	Returns the base of natural logarithms	2.0	3.0	1.0
LN2	Returns the natural logarithm of 2	2.0	3.0	1.0
LN10	Returns the natural logarithm of 10	2.0	3.0	1.0
LOG2E	Returns the base-2 logarithm of E	2.0	3.0	1.0
LOG10E	Returns the base-10 logarithm of E	2.0	3.0	1.0
PI	Returns PI	2.0	3.0	1.0
SQRT1_2	Returns the square root of 0.5	2.0	3.0	1.0
SQRT2	Returns the square root of 2	2.0	3.0	1.0

Methods	Explanation	NN	IE	ECMA
abs()	Returns the absolute value a number	2.0	3.0	1.0
acos()	Returns the arccosine of a number	2.0	3.0	1.0
asin()	Returns the arcsine of a number	2.0	3.0	1.0
atan()	Returns the arctangent of a number	2.0	3.0	1.0
atan2()	Returns the angle from the x axis to a point	2.0	3.0	1.0
ceil()	Returns a the nearest whole number greater than or equal to the number	2.0	3.0	1.0
cos()	Returns the cosine of a number	2.0	3.0	1.0
exp()	Returns the base of logarithms raised to a power	2.0	3.0	1.0

floor()	Returns a the nearest whole number less than or equal to the number	2.0	3.0	1.0
log()	Returns the logarithm of a number	2.0	3.0	1.0
max()	Returns the number with the highest value of two numbers	2.0	3.0	1.0
min()	Returns the number with the lowest value of two numbers	2.0	3.0	1.0
pow()	Returns the value of a base expression taken to a specified power	2.0	3.0	1.0
random()	Returns a random number	2.0	3.0	1.0
round()	Returns a number rounded to the nearest whole number	2.0	3.0	1.0
sin()	Returns the sine of a number	2.0	3.0	1.0
sqrt()	Returns the square root of a number	2.0	3.0	1.0
tan()	Returns the tangent of a number	2.0	3.0	1.0

HTML:

Do you want to try it?

If you are running Windows, start Notepad and type in the following text:

```
<html>
<head>
<title>Title of page</title>
</head>
<body>
This is my first homepage. <b>This is some text</b>
</body>
</html>
```

Save the file as "c:\mypage.htm".

Start your Internet browser and type "c:\mypage.htm" in the browser's address field, and the browser will display the page.

Basic Tags:

Start Tag	NN	IE	W3	Purpose
<html>	3.0	3.0	3.2	Defines a html document
<body>	3.0	3.0	3.2	Defines the documents' body
<h1>-<h6>	3.0	3.0	3.2	Defines header 1 to header 6
<p>	3.0	3.0	3.2	Defines a paragraph
<br>	3.0	3.0	3.2	Inserts a single line break
<hr>	3.0	3.0	3.2	Defines a horizontal rule
<!-->	3.0	3.0	3.2	Defines a comment in the HTML source code

Text Formatting Tags:

Start Tag	NN	IE	W3	Purpose
<b>	3.0	3.0	3.2	Defines bold text

<big>	3.0	3.0	3.2	Defines big text
<em>	3.0	3.0	3.2	Defines emphasized text
<i>	3.0	3.0	3.2	Defines italic text
<small>	3.0	3.0	3.2	Defines small text
<strong>	3.0	3.0	3.2	Defines strong text
<sub>	3.0	3.0	3.2	Defines subscripted text
<sup>	3.0	3.0	3.2	Defines superscripted text
<ins>		4.0	4.0;	Defines inserted text
<del>		4.0	4.0	Defines deleted text
<s>				Deprecated. Use <del> instead
<strike>				Deprecated. Use <del> instead
<u>				Deprecated. Use styles instead

Computer Output Tags:

Start Tag	NN	IE	W3	Purpose
<code>	3.0	3.0	3.2	Defines computer code text
<kbd>	3.0	3.0	3.2	Defines keyboard text
<samp>	3.0	3.0	3.2	Defines sample computer code
<tt>	3.0	3.0	3.2	Defines teletype text
<var>	3.0	3.0	3.2	Defines a variable
<pre>	3.0	3.0	3.0	Defines preformatted text
<listing>				Deprecated. Use <pre> instead
<plaintext>				Deprecated. Use <pre> instead
<xmp>				Deprecated. Use <pre> instead

Citations, Quotations, Definition Tags:

Start Tag	NN	IE	W3	Purpose
<abbr>			4.0	Defines an abbreviation
<acronym>			4.0	Defines an acronym
<address>	4.0	4.0	4.0	Defines an address element
<bdo>			4.0	Defines the text direction
<blockquote>	3.0	3.0	3.2	Defines a long quotation
<q>			4.0	Defines a short quotation
<cite>	3.0	3.0	3.2	Defines a citation
<dfn>		3.0	3.2	Defines a definition term

Commonly Used Character Entities

Result	Description	Entity Name	Entity Number
©	copyright	&copy;	&#169;
®	registered trademark	&reg;	&#174;
™	trademark		&#8482;
	non-breaking space	&nbsp;	&#161;

&	ampersand	&amp;	&#38;
«	angle quotation mark (left)	&laquo;	&#171;
»	angle quotation mark (right)	&raquo;	&#187;
"	quotation mark	&quot;	&#34;
<	less than	&lt;	&#60;
>	greater than	&gt;	&#61;
×	multiplication	&times;	&#215;
÷	division	&divide;	&#247;

Link Tags:

Start Tag	NN	IE	W3	Purpose
<a>	3.0	3.0	3.2	Defines an anchor

Link nicht unterstrichen:

```
<html>
<body>

<a href="http://www.w3schools.com"
style="text-decoration:none">
THIS IS A LINK!
</a>

</body>
</html>
```

Link a Location on the same place:

```
<html>
<body>

<p>
<a href="#C4">
See also Chapter 4.
</a>
</p>

<p>
<h2>Chapter 1</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 2</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 3</h2>
<p>This chapter explains ba bla bla</p>

<a name="C4"></a>
<h2>Chapter 4</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 5</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 6</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 7</h2>
```

```
<p>This chapter explains ba bla bla</p>

<h2>Chapter 8</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 9</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 10</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 11</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 12</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 13</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 14</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 15</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 16</h2>
<p>This chapter explains ba bla bla</p>

<h2>Chapter 17</h2>
<p>This chapter explains ba bla bla</p>

</body>
</html>
```

Open link in a new window:

```
<html>
<body>

<a href="http://www.microsoft.com"
target="_blank">Microsoft</a>

<p>
If you set the target attribute of a link to "_blank",
the link will open in a new window.
</p>

</body>
</html>
```

Mailto:

```
<html>

<body>

<a href="mailto:xxx@yyy.com?Subject=Hello again">
Send e-mail!
</a>

</body>
</html>
```

Frame Tags:

Start Tag	NN	IE	W3	Purpose
<frameset>	3.0	3.0	4.0	Defines a set of frames
<frame>	3.0	3.0	4.0	Defines a sub window (a frame)
<noframes>	3.0	3.0	4.0	Defines a noframe section for browsers that do not handle frames
<iframe>		3.0	4.0	Defines an inline sub window (frame)

Table Tags:

Start Tag	NN	IE	W3	Purpose
<table>	3.0	3.0	3.2	Defines a table
<th>	3.0	3.0	3.2	Defines a table header
<tr>	3.0	3.0	3.2	Defines a table row
<td>	3.0	3.0	3.0	Defines a table cell
<caption>	3.0	3.0	3.2	Defines a table caption
<colgroup>		3.0	4.0	Defines groups of table columns
<col>		3.0	4.0	Defines the attribute values for one or more columns in a table
<thead>		4.0	4.0	Defines a table header that will not scroll
<tbody>		4.0	4.0	Defines a table body that scrolls within a fixed table header and table footer
<tfoot>		4.0	4.0	Defines a table footer that will not scroll

List Tags:

Start Tag	NN	IE	W3	Purpose
<ol>	3.0	3.0	3.2	Defines an ordered list
<ul>	3.0	3.0	3.2	Defines an unordered list
<li>	3.0	3.0	3.2	Defines a list item
<dl>	3.0	3.0	3.2	Defines a definition list
<dt>	3.0	3.0	3.2	Defines a definition term
<dd>	3.0	3.0	3.2	Defines a definition description
<dir>				Deprecated. Use <ul> instead
<menu>				Deprecated. Use <ul> instead

Form Tags:

Start Tag	NN	IE	W3	Purpose
<form>	3.0	3.0	3.2	Defines a form for user input
<input>	3.0	3.0	3.2	Defines an input field
<textarea>	3.0	3.0	3.2	Defines a text-area (a multi-line text input control)
<label>			4.0	Defines a label to a control
<fieldset>		4.0	4.0	Defines a fieldset
<legend>		4.0	4.0	Defines a caption for a fieldset
<select>	3.0	3.0	3.2	Defines a selectable list (a drop-down box)

<optgroup>			4.0	Defines an option group
<option>	3.0	3.0	3.2	Defines an option in the drop-down box
<button>		4.0	4.0	Defines a push button
<isindex>				Deprecated. Use <input> instead

Image Tags:

Start Tag	NN	IE	W3	Purpose
<img>	3.0	3.0	3.2	Defines an image
<map>	3.0	3.0	3.2	Defines an image map
<area>	3.0	3.0	3.2	Defines an area inside an image map

Using the HTML <font> Tag

With HTML code like this, you can specify both the size and the type of the browser output :

```
<p><font size="2" face="Verdana">
This is a paragraph.
</font></p>
<p><font size="3" face="Times">
This is another paragraph.
</p>
```

Try it yourself

The <font> tag should not be used

The <font> tag is deprecated in the latest versions of HTML (HTML 4 and XHTML).

The World Wide Web Consortium (W3C) has removed the <font> tag from its recommendations. If future versions of HTML, style sheets (CSS) will be used to define the layout and display properties of HTML elements.

You can read more about this in the next chapters.

Style Tags

Start Tag	NN	IE	W3	Purpose
<style>	4.0	3.0	3.2	Defines a style in a document
<link>	4.0	3.0	3.2	Defines the relationship between two linked documents
<div>	3.0	3.0	3.2	Defines a division/section in a document
<span>		3.0	4.0	Defines a section in a document
<font>				Deprecated. Use styles instead
<basefont>				Deprecated. Use styles instead
<center>				Deprecated. Use styles instead

Information inside the Head Element

The elements inside the head should not be displayed by a browser. According to the HTML standard, only a few tags are legal inside the head section. These are: <base>, <link>, <meta>, <title>, <style>, and <script>.

Look at the following illegal construct:

```
<head>
  <p>This is some text</p>
</head>
```

In this case the browser has two options:

- Display the text because it is inside a paragraph element
- Hide the text because it is inside a head element

If you put an HTML element like <h1> or <p> inside a head element like this, most browsers will display it, even if it is illegal.

Should browsers forgive you for errors like this? We don't think so. Others do.

Head Tags:

Start Tag	NN	IE	W3	Purpose
<head>	3.0	3.0	3.2	Defines information about the document
<title>	3.0	3.0	3.2	Defines the document title
<base>	3.0	3.0	3.2	Defines a default reference to external resources
<link>	4.0	3.0	3.2	Defines the relationship between two linked documents
<meta>	3.0	3.0	3.2	Defines meta information

Start Tag	NN	IE	W3	Purpose
<!doctype>			3.2	Defines the document type. This tag goes before the <html> start tag.

Keywords for Search Engines

Some of the search engines on the World Wide Web, will use the description or the keyword attribute of your meta tags to index your pages.

This meta element defines a description of your page:

```
<meta name="description" content="Free Web tutorials on HTML, CSS, XML, and XHTML">
```

This meta element defines keywords for your page:

```
<meta name="keywords" content="HTML, DHTML, CSS, XML, XHTML, JavaScript, VBScript">
```

Script Tags:

Start Tag	NN	IE	W3	Purpose
<script>	3.0	3.0	3.2	Defines a script
<noscript>	3.0	3.0	4.0	Defines a noscript section for browsers that do not support scripting
<object>		3.0	4.0	Defines an embedded object
<param>	3.0	3.0	3.2	Defines run-time settings (parameters) for an object
<applet>				Deprecated. Use <object> instead

List of differences:

- Tags and attributes must be in lowercase
- All XHTML elements must be closed
- Attribute values must be quoted and minimization is forbidden
- The id attribute replaces the name attribute
- The script element needs a type definition
- Documents must conform to XML rules
- XHTML documents have some mandatory elements

Tags and attributes must be in lower case

This is because XHTML documents are XML applications. XML is case-sensitive. Tags like `<br>` and `<BR>` are interpreted as different tags.

This is wrong:

```
<BODY BGCOLOR="#CCCCCC">
<P>This is a paragraph<P>
</BODY>
```

This is correct:

```
<body bgcolor="#CCCCCC">
<p>This is a paragraph</p>
</body>
```

All XHTML elements must be closed

Non-empty elements must have an end tag.

This is wrong:

```
<p>This is a paragraph
<p>This is another paragraph
```

This is correct:

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

Empty elements must either have an end tag or the start tag must end with />.

This is wrong:

```
This is a break<br>
Here comes a horizontal rule:<hr>
```

This is correct:

```
This is a break<br/>
This is a break too<br></br>

Here comes a horizontal rule:<hr/>
```

IMPORTANT Compatibility Note:

To make your XHTML compatible with today's browsers, you should add an extra space before the backslash symbol like this: <br /> and this <hr />.

Attribute values must be quoted

This is wrong:

```
<table rows=3>
```

This is correct:

```
<table rows="3">
```

Attribute minimization is forbidden

This is wrong:

```
<dl compact>
<input checked>
<option selected>
```

This is correct:

```
<dl compact="compact">
<input checked="checked">
<option selected="selected">
```

The id attribute replaces the name attribute

HTML 4.0 defined the name attribute for the elements a, applet, frame, iframe, img, and map. In XHTML the name attribute is deprecated. Use id instead.

This is wrong:

```
<img src="picture.gif" name="picture1"/>
```

This is correct:

```
<img src="picture.gif" id="picture1"/>
```

(To interoperate with older browsers for a while, you can try to use both name and id, with identical attribute values).

IMPORTANT Compatibility Note:

To make your XHTML compatible with today's browsers, you should add an extra space before the "/" symbol like this:

```
<img src="picture.gif" id="picture1" />
```

Script and Style elements

In XHTML the characters like "<" and "&" are treated as a part of the markup. To avoid this inside scripts or other elements, wrap the content with a CDATA definition like this:

```
<script>
  <![CDATA[
  ... script content ...
  ]]>
</script>
```

Documents must conform to XML rules

Since XHTML documents are XML applications, XHTML documents must conform to the following XML rules:

Elements must be properly nested

In HTML some elements can be improperly nested within each other like this:

```
<b><i>This text is bold and italic</b></i>
```

In XHTML all elements must be properly nested within each other like this

```
<b><i>This text is bold and italic</i></b>
```

Documents must be well-formed

All XHTML elements must be nested within the <html> root element. All other elements can have sub (children) elements. Sub elements must be in pairs and correctly nested within their parent element. The basic document structure is:

```
<html>
<head> ... </head>
<body> ... </body>
</html>
```

You can read more about the XML syntax in our XML School.

XHTML documents have some mandatory elements

All XHTML documents must have a DOCTYPE declaration. The html, head and body elements must be present, and the title must be present in the head element.

This is a minimum XHTML document template::

```
<!DOCTYPE html>
<html>
<head>
<title>This is a Title</title>
</head>
<body>
.
Body text goes here
.
</body>
</html>
```

An XHTML document consists of three main parts:

- the DOCTYPE
- the Head
- the Body

The basic document structure is:

```
<!DOCTYPE ...>
<html  ... >
<head> ... </head>
<body> ... </body>
</html>
```

This is a simple XHTML document:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/strict.dtd">
<html>
<head>
<title>simple document</title>
</head>
<body>
<p>a simple paragraph</p>
</body>
</html>
```

The DOCTYPE definition specifies the document type:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/strict.dtd">
```

The rest of the document looks like HTML:

```
<html>
<head>
<title>simple document</title>
</head>
<body>
<p>a simple paragraph</p>
</body>
</html>
```

Document Type Definitions

- DTD specifies the syntax of a web page in SGML.
- DTD is used by SGML applications, such as HTML, to specify rules that apply to the markup of documents of a particular type, including a set of element and entity declarations.
- XHTML is specified in an SGML document type definition or 'DTD'.
- An XHTML DTD describes in precise, computer-readable language the allowed syntax and grammar of XHTML markup.

Validating an XHTML document's content involves checking its markup against a DTD and reporting markup errors. **There are currently 3 XHTML document types:**

- STRICT
- TRANSITIONAL
- FRAMESET

XHTML 1.0 specifies three XML document types that correspond to the three HTML 4.0 DTDs: Strict, Transitional, and Frameset.

XHTML 1.0 Strict

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/strict.dtd">
```

Use this when you want really clean markup, free of presentational clutter. Use this together with Cascading Style Sheets.

XHTML 1.0 Transitional

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/transitional.dtd">
```

Use this when you need to take advantage of HTML's presentational features because many of your readers don't have the latest browsers that understand Cascading Style Sheets.

XHTML 1.0 Frameset

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/frameset.dtd">
```

Use this when you want to use HTML Frames to partition the browser window into two or more frames.

How to put a VBScript into an HTML document

```
<html>
<head>
</head>
<body>
<script language="VBScript">
document.write("I am written using VBScript!")
</script>
</body>
</html>
```

And it produces this output:

```
I am written using VBScript!
```

To insert a script in an HTML document, use the <script> tag. Use the language attribute to define the scripting language.

```
<script language="VBScript">
```

Then comes the VBScript: The command for writing some text on a page is **document.write**:

```
document.write("I am written using VBScript!")
```

The script ends:

```
</script>
```

How to handle older browsers

Older browsers that does not support scripts will display the script as page content. To prevent them from doing this, you can use the HTML comment tag:

```
<script language="VBScript">
<!--
    some statements
-->
</script>
```

Where to put the VBScript

Scripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user trigger an event.

Scripts in the head section: Scripts to be executed when they are called or when an event is triggered, goes in the head section. When you place a script in the head section you will assure that the script is loaded before anyone uses it.

```
<html>
<head>
<script language="VBScript">
     some statements
</script>
</head>
```

Scripts in the body section: Scripts to be executed when the page loads, goes in the body section. When you place a script in the body section it generates the content of the page.

```
<html>
<head>
</head>
<body>
<script language="VBScript">
     some statements
</script>
</body>
```

Scripts in both the body and the head section: You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

```
<html>
<head>
<script language="VBScript">
     some statements

</head>
<body>
<script language="VBScript">
     some statements
</script>
</body>
```

What is a Variable?

A variable is a "container" for information you want to store. A variables value can change during the script. You can refer to a variable by name to see its value or to change its value. In VBScript, all variables are of type *variant*, that can store different types of data.

Rules for Variable names:

- Must begin with a letter
- Can not contain a period (.)
- Must not exceed 255 characters

Declaring Variables

You can declare variables with the Dim, Public or the Private statement. Like this:

```
dim name
name = some value
```

Now you have created a variable. The name of the variable is "name".

You can also declare variables by using its name in your script. Like this:

```
name = some value
```

Now you have also created a variable. The name of the variable is "name".

However, the last method is not a good practice, because you can misspell the variable name later in your script, and that can cause strange results when your script is running. This is because when you misspell for example the "name" variable to "nime" the script will automatically create a new variable called "nime". To prevent your script from doing this you can use the Option Explicit statement. When you use this statement you will have to declare all your variables with the dim, public or private statement. Put the Option Explicit statement on the top of your script. Like this:

```
option explicit
dim name
```

Assigning Values to Variables

You assign a value to a variable like this:

```
name = "Hege"
i = 200
```

The variable name is on the left side of the expression and the value you want to assign to the variable is on the right. Now the variable "name" has the value "Hege".

Lifetime of Variables

How long a variable exists is its lifetime.

When you declare a variable within a procedure, only code within that procedure can access or change the value of that variable. When the procedure exits, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different procedures, because each is recognized only by the procedure in which it is declared.

If you declare a variable outside a procedure, all the procedures in your script will recognize it. These variables exists from the time they are declared until the time the script is finished running.

Array Variables

Some times you want to assign more than one value to a single variable. Then you can create a variable that can contain a series of values. This is called an array variable. The declaration of an array variable uses parentheses ( ) following the variable name. In the following example, an array containing 3 elements is declared:

```
dim names(2)
```

The number shown in the parentheses is 2. We start at zero so this array contains 3 elements. This is a fixed-size array. You assign data to each of the elements of the array like this:

```
names(0) = "Tove"
names(1) = "Jani"
names(2) = "Ståle"
```

Similarly, the data can be retrieved from any element using an index into the particular array element you want. Like this:

```
mother = names(0)
```

You can have up to 60 dimensions in an array. Multiple dimensions are declared by separating the numbers in the parentheses with commas. Here we have a two-dimensional array consisting of 5 rows and 7 columns:

```
dim table(4, 6)
```

VBScript Procedures

We have two kinds of procedures: The Sub procedure and the Function procedure.

A Sub Procedure:

- Is a series of statements, enclosed by the Sub and End Sub statements
- Perform actions, but do not return a value
- Can take arguments that are passed to it by a calling procedure
- Without arguments, must include an empty set of parentheses ()

```
Sub mysub()
        some statements
End Sub
```

The Function Procedure

- Is a series of statements, enclosed by the Function and End Function statements
- Perform actions, but can also return a value
- Can take arguments that are passed to it by a calling procedure
- Without arguments, must include an empty set of parentheses ()
- Returns a value by assigning a value to its name

```
Function myfunction()
        some statements
        myfunction = some value
```

Using Sub and Function Procedures in Code

When you call a Function in your code, you do like this:

```
name = findname()
```

Here you call a Function called "findname", the Function returns a value that will be stored in the variable "name".

Or, you can do like this:

```
msgbox "Your name is " & findname()
```

Here you also call a Function called "findname", the Function returns a value that will be displayed in the message box.

When you call a Sub procedure you can just type the name of the procedure. You can use the Call statement, like this:

```
Call MyProc(argument)
```

Or, you can omit the call statement, like this:

```
MyProc argument
```

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 VBScript we have two conditional statements:

- **If...Then...Else statement** - use this statement if you want to select one of two sets of lines to execute
- **Select Case statement** - use this statement if you want to select one of many sets of lines to execute

---

If....then.....else

You should use the IF statement if you want to execute some code if a condition is true, or if you want to select one of two blocks of code to execute.

If you want to execute only one statement when a condition is true, use this syntax for the if...then...else statement, like this:

```
If i = 10 Then msgbox "Hello"
```

Notice that there is no ..else.. in this syntax. You just tell the code to perform **one action** if the condition (i) is equal to some value (in this case the value is 10).

If you want to execute more than one statement when a condition is true, use this syntax for the if...then...else statement, like this:

```
If i = 10 Then
    msgbox "Hello"
    i = 11
    more statements
End If
```

There is no ..else.. in this syntax either. You just tell the code to perform **multiple actions** if the condition (i) is equal to some value (in this case the value is 10).

If you want to execute some statements if a condition is true and execute others if a condition is false, use this syntax for the if...then...else statement, like this:

```
If i = 10 Then
    msgbox "Hello"
    i = 11
Else
    msgbox "Goodbye"
End If
```

The first block of code will be executed if the condition is true (if i is equal to 10), the other block will be executed if the condition is false (if i is not equal to 10).

Select case

You should use the SELECT statement if you want to select one of many blocks of code to execute.

```
Select Case payment
    Case "Cash"
        msgbox "You are going to pay cash"
    Case "Visa"
        msgbox "You are going to pay with visa"
    Case Else
        msgbox "Unknown method of payment"
End Select
```

This is how it works: First we have a single expression (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each Case in the structure. If there is a match, the block of code associated with that Case is executed.

Looping Statements

Very often when you write code, you want allow the same block of code to run a number of times. You can use looping statements in your code to do this.

In VBScript we have four looping statements:

- **Do...Loop statement** - loops while or until a condition is true
- **While...Wend statement** - use Do...Loop instead
- **For...Next statement** - run statements a specified number of times.
- **For Each...Next statement** - run statements for each item in a collection or each element of an array

---

Do...Loop

You can use Do...Loop statements to run a block of code when you do not know how many repetitions you want. The block of code are repeated while a condition is true or until a condition becomes true.

Repeating Code While a Condition is True

You use the While keyword to check a condition in a Do...Loop statement.

```
Do While i > 10
    some code
Loop
```

Notice that if i is for example 9, the code inside the loop will never be executed.

```
Do
    some code
Loop While i > 10
```

Notice that in this example the code inside this loop will be executed at least one time, even if i is less than 10.

Repeating Code Until a Condition Becomes True

You use the Until keyword to check a condition in a Do...Loop statement.

```
Do Until i = 10
     some code
Loop
```

Notice that if i is equal to 10, the code inside the loop will never be executed.

```
Do
     some code
Loop Until i = 10
```

Notice that in this example the code inside this loop will be executed at least one time, even if i is equal to 10.

Exiting a Do...Loop

You can exit a Do...Loop statement with the Exit Do keyword.

```
Do Until i = 10
    i = i - 1
    If i < 10 Then Exit Do
Loop
```

Notice that the code inside this loop will be executed as long i is different from 10, and as long as i is greater than 10.

For...Next

You can use For...Next statements to run a block of code when you know how many repetitions you want.

You can use a counter variable that increases or decreases with each repetition of the loop, like this:

```
For i = 1 to 10
    some code
Next
```

The For statement specifies the counter variable i and its start and end values. The Next statement increases i by 1.

Using the Step keyword, you can increase or decrease the counter variable by the value you specify.

```
For i = 2 To 10 Step 2
    some code
Next
```

In the example above, i is increased by 2 each time the loop repeats. When the loop is finished, total is the sum of 2, 4, 6, 8, and 10.

To decrease the counter variable, you use a negative Step value. You must specify an end value that is less than the start value.

```
For i = 10 To 2 Step -2
    some code
Next
```

In the example above, i is decreased by 2 each time the loop repeats. When the loop is finished, total is the sum of 10, 8, 6, 4, and 2.

Exiting a For...Next

You can exit a For...Next statement with the Exit For keyword.

For Each...Next

A For Each...Next loop repeats a block of code for each item in a collection, or for each element of an array.

The For Each...Next statement looks almost identical to the For...Next statement. The difference is that you do not have to specify the number of items you want to loop through.

```
dim names(3)
names(0) = "Tove"
names(1) = "Jani"
names(2) = "Hege"

For Each item in names
    some code
Next
```