9. JavaScript Functions

A function is a block of code that will be executed when "someone" calls it:

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction()
{
alert("Hello World!");
}
</script>
</head>

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

9.1. JavaScript Function Syntax

A function is written as a code block (inside curly { } braces), preceded by the **function** keyword:

```
function functionname()
{
some code to be executed
}
```

The code inside the function will be executed when "someone" calls the function.

The function can be called directly when an event occurs (like when a user clicks a button), and it can be called from "anywhere" by JavaScript code.

JavaScript is case sensitive. The function keyword must be written in lowercase letters, and the function must be called with the same capitals as used in the function name.

9.2. Calling a Function with Arguments

When you call a function, you can pass along some values to it, these values are called *arguments* or *parameters*.

These arguments can be used inside the function.

You can send as many arguments as you like, separated by commas (,)

```
myFunction(argument1,argument2)
```

Declare the argument, as variables, when you declare the function:

```
function myFunction(var1,var2)
{
some code
}
```

The variables and the arguments must be in the expected order. The first variable is given the value of the first passed argument etc.

Example

```
<button onclick="myFunction('Harry
Potter','Wizard')">Try it</button>

<script>
function myFunction(name,job)
{
alert("Welcome " + name + ", the " + job);
}
</script>
```

The function above will alert "Welcome Harry Potter, the Wizard" when the button is clicked.

The function is flexible, you can call the function using different arguments, and different welcome messages will be given:

Example

```
<button onclick="myFunction('Harry Potter','Wizard')">Try
it</button>
```

```
<button onclick="myFunction('Bob','Builder')">Try
it</button>
```

The example above will alert "Welcome Harry Potter, the Wizard" or "Welcome Bob, the Builder" depending on which button is clicked.

9.3. Functions With a Return Value

Sometimes you want your function to return a value back to where the call was made.

This is possible by using the *return* statement.

When using the *return* statement, the function will stop executing, and return the specified value.

Syntax

```
function myFunction()
{
var x=5;
return x;
}
```

The function above will return the value 5.

Note: It is not the entire JavaScript that will stop executing, only the function. JavaScript will continue executing code, where the function-call was made from.

The function-call will be replaced with the return value:

```
var myVar=myFunction();
```

The variable myVar holds the value 5, which is what the function "myFunction()" returns.

You can also use the return value without storing it as a variable:

```
document.getElementById("demo").innerHTML=myFunction();
```

The innerHTML of the "demo" element will be 5, which is what the function "myFunction()" returns.

You can make a return value based on arguments passed into the function:

Example

Calculate the product of two numbers, and return the result:

```
function myFunction(a,b)
{
return a*b;
}
document.getElementById("demo").innerHTML=myFunction(4,3);
```

The innerHTML of the "demo" element will be:

```
12
```

The return statement is also used when you simply want to exit a function. The return *value* is optional:

```
function myFunction(a,b)
{
if (a>b)
  {
  return;
  }
x=a+b
}
```

The function above will exit the function if a>b, and will not calculate the sum of a and b.

9.4. Local JavaScript Variables

A variable declared (using var) within a JavaScript function becomes **LOCAL** and can only be accessed from within that function. (the variable has local scope).

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

Local variables are deleted as soon as the function is completed.

9.5. Global JavaScript Variables

Variables declared outside a function, become **GLOBAL**, and all scripts and functions on the web page can access it.

9.6. The Lifetime of JavaScript Variables

The lifetime JavaScript variables starts when they are declared.

Local variables are deleted when the function is completed.

Global variables are deleted when you close the page.

9.7. Assigning Values to Undeclared JavaScript Variables

If you assign a value to variable that has not yet been declared, the variable will automatically be declared as a **GLOBAL** variable.

This statement:

```
        carname="Volvo";
```

will declare the variable *carname* as a global variable , even if it is executed inside a function.