JavaScript

Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro
Prof. Edwar Saliba Júnior

JS Tutorial

Introdução

- É conhecida como “A Linguagem Script da WEB”;
- É utilizada em bilhões de páginas para:
  - adicionar funcionalidades,
  - validar formulários,
  - comunicação com o servidor e
  - muito mais;
- É muito simples de ser aprendida.

Características

- É uma linguagem de programação leve;
- O código gerado em JavaScript pode ser inserido em páginas HTML;
- O código JavaScript inserido em páginas HTML pode ser executado por todos os navegadores modernos;
- É uma linguagem muito simples de ser aprendida.

Observação

- **JavaScript** e **Java**, apesar da semelhança do nome, são linguagens totalmente distintas uma da outra.

Como Fazer

Scripts in HTML must be inserted between <script> and </script> tags.

Scripts 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 the <script> and </script> contain the JavaScript:

```
<script>
alert("My First JavaScript");
</script>
```

You don't have to understand the code above. Just take it for a fact, that the browser will interpret and execute the JavaScript code between the <script> and </script> tags.

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 em <body>

JavaScript in <body>

In this example, JavaScript writes into the HTML <body> while the page loads:

Example

```
<!DOCTYPE html>
<html>
<body>
.
.
<script>
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph</p>");
</script>
.
.
</body>
</html>
```

JavaScript em <body> Exemplo

Source Code: Submit Code

```
<!DOCTYPE html>
<html>
  <body>
    <p>
      JavaScript can write directly into the HTML output stream:
    </p>

    <script>
      document.write("<h1>This is a heading</h1>");
      document.write("<p>This is a paragraph.</p>");
    </script>

    <p>
     You can only use <strong>document.write</strong> in the HTML
output. If you use it after the document has loaded (e.g. in a
function), the whole document will be overwritten.
    </p>
  </body>
</html>
```

Result:

JavaScript can write directly into the HTML output stream:

This is a heading

This is a paragraph.

You can only use **document.write** in the HTML output. If you use it after the document has loaded (e.g. in a function), the whole document will be overwritten.

JavaScript Function em <body>

A JavaScript Function in <head>

In this example, a JavaScript function is placed in the <head> section of an HTML page.

The function is called when a button is clicked:

Example

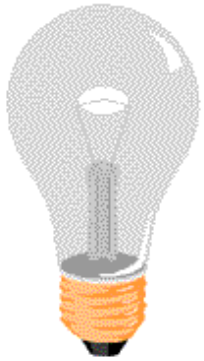
```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction()
{
document.getElementById("demo").innerHTML="My First JavaScript Function";
}
</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 Function em <body> Exemplo 2

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      function changeImage(){
        element=document.getElementById('myimage')
        if (element.src.match("bulbon")){
          element.src="pic_bulboff.gif";
        }
        else{
          element.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>
```

Result:



Click the light bulb to turn on/off the light

Result:

Click the light bulb to turn on/off the light

JavaScript em Arquivo

External JavaScripts

Scripts can also be placed in external files. External files often contain code to be used by several different web pages.

External JavaScript files have the file extension .js.

To use an external script, point to the .js file in the "src" attribute of the <script> tag:

Example

```
<!DOCTYPE html>
<html>
<body>
<script src="myScript.js"></script>
</body>
</html>
```

You can place the script in the <head> or <body> as you like. The script will behave as if it was located exactly where you put the <script> tag in the document.

External scripts cannot contain <script> tags.

JavaScript em Arquivo Exemplo

Source Code: Submit Code

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My Web Page</h1>

    <p id="demo">A Paragraph.</p>

    <button type="button" onclick="myFunction()">Try it</button>

    <p><strong>Note:</strong> myFunction is stored in an external
file called "myScript.js".</p>

    <script type="text/javascript" src="myScript.js"></script>

    </body>
</html>
```

Result:

Após apertar o botão "*Try it*".

Result:

Manipulando Elementos HTML

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:

Example

Access the HTML element with the specified id, and change its content:

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>

<p id="demo">My First Paragraph</p>

<script>
document.getElementById("demo").innerHTML="My First JavaScript";
</script>

</body>
</html>
```

The JavaScript is executed by the web browser. In this case, the browser will access the HTML element with id="demo", and replace its content (innerHTML) with "My First JavaScript".

Manipulando Elementos HTML Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My First Web Page</h1>

    <p id="demo">My First Paragraph.</p>

    <script>
      document.getElementById("demo").innerHTML="My First JavaScript";
    </script>
  </body>
</html>
```

Result:

My First Web Page

My First JavaScript

Escrevendo no Documento

Writing to The Document Output

The example below writes a <p> element directly into the HTML document output:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>

<script>
document.write("<p>My First JavaScript</p>");
</script>

</body>
</html>
```

Escrevendo no Documento Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My First Web Page</h1>

    <script>
      document.write("<p>My First JavaScript</p>");
    </script>
  </body>
</html>
```

Result:

My First Web Page

My First JavaScript

Advertência

Warning

Use document.write() only to write directly into the document output.

If you execute document.write after the document has finished loading, the entire HTML page 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("Oops! The document disappeared!");
}
</script>

</body>
</html>
```

Advertência
Exemplo

Source Code:

```
<!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("Oops! The document disappeared!");
      }
    </script>
  </body>
</html>
```

Result:

My First Web Page

My First Paragraph.

Try it

Result:

Oops! The document disappeared!

Após apertar o botão "*Try it*".

Declarações

- JavaScript é um conjunto de declarações a serem executadas pelo navegador.

JavaScript Statements

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 with id="demo":

```
document.getElementById("demo").innerHTML="Hello Dolly";
```

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.

You might see examples without semicolons.
Ending statements with semicolon is optional in JavaScript.

Código em 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 HTML elements:

Example

```
document.getElementById("demo").innerHTML="Hello Dolly";
document.getElementById("myDIV").innerHTML="How are you?";
```

Código em JavaScript Exemplo

Source Code: Submit Coc

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My Web Page</h1>

    <p id="demo">A Paragraph.</p>

    <div id="myDIV">A DIV.</div>

    <script>
      document.getElementById("demo").innerHTML="Hello Dolly";
      document.getElementById("myDIV").innerHTML="How are you?";
    </script>
  </body>
</html>
```

Result:

My Web Page

Hello Dolly

How are you?

Blocos de Comandos

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 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?";
}
```

You will learn more about functions in later chapters.

Blocos de Comandos Exemplo

Source Code: Submit Code »

```
<!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>
```

Result:

My Web Page

I am a paragraph.

I am a div.

Try it

When you click on "Try it", the two elements will change.

Result:

My Web Page

Hello Dolly

How are you?

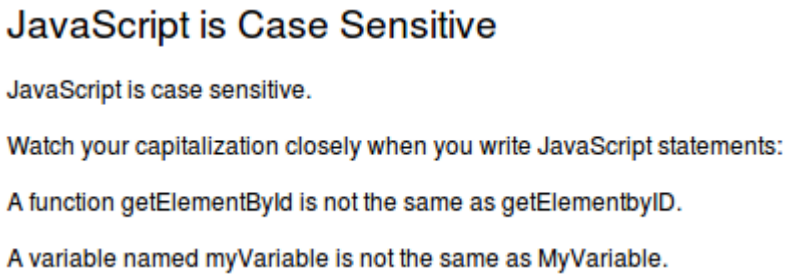
Try it

When you click on "Try it", the two elements will change.

Após apertar o botão "*Try it*".

JavaScript é *Case Sensitive*

- Como na linguagem C ou Java, o JavaScript também é sensível a caixa. Assim sendo, preste bastante atenção ao escrever os comandos.



JavaScript is Case Sensitive

JavaScript is case sensitive.

Watch your capitalization closely when you write JavaScript statements:

A function getElementById is not the same as getElementbyID.

A variable named myVariable is not the same as MyVariable.

Atenção

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 name="Hege";
var name = "Hege";
```

Break up a Code Line

You can break up a code line **within a text string** with a backslash. The example below will be displayed properly:

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

However, you cannot break up a code line like this:

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

Did You Know?

JavaScript is a scripting language. Scripting code is executed line by line while the browser reads it. With traditional programming, all the code has to be compiled before it can be executed.

Comentário em JavaScript

JavaScript comments can be used to make the code more readable.

JavaScript Comments

Comments will not be executed by JavaScript.

Comments can be added to explain the JavaScript, or to make the code more readable.

Single line comments start with //.

The following example uses single line comments to explain the code:

Example

```
// Write to a heading:
document.getElementById("myH1").innerHTML="Welcome to my Homepage";
// Write to a paragraph:
document.getElementById("myP").innerHTML="This is my first paragraph.";
```

Comentário em JavaScript Exemplo

Source Code: Submit Cod

```
<!DOCTYPE html>
<html>
  <body>
    <h1 id="myH1"></h1>

    <p id="myP"></p>

    <script>
      // Write to a heading:
      document.getElementById("myH1").innerHTML="Welcome to my
Homepage";

      // Write to a paragraph:
      document.getElementById("myP").innerHTML="This is my first
paragraph.";
    </script>

    <p><strong>Note:</strong> The comments are not executed.</p>
  </body>
</html>
```

Result:

Welcome to my Homepage

This is my first paragraph.

Note: The comments are not executed.

Comentário em Várias Linhas

JavaScript Multi-Line Comments

Multi line comments start with /* and end with */.

The following example uses a multi line comment to explain the code:

Example

```
/*
The code below will write
to a heading and to a paragraph,
and will represent the start of
my homepage:
*/
document.getElementById("myH1").innerHTML="Welcome to my Homepage";
document.getElementById("myP").innerHTML="This is my first paragraph.";
```

Comentário em Várias Linhas Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <h1 id="myH1"></h1>
    <p id="myP"></p>

    <script>
      /* The code below will write to a heading and to a paragraph,
         and will represent the start of my homepage:
      */
      document.getElementById("myH1").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>
```

Result:

Welcome to my Homepage

This is my first paragraph.

Note: The comment-block is not executed.

Usando Comentários para Prevenir Execuções – Parte 1

Using Comments to Prevent Execution

In the following example the comment is used to prevent the execution of one of the codelines (can be suitable for debugging):

Example

```
//document.getElementById("myH1").innerHTML="Welcome to my Homepage";
document.getElementById("myP").innerHTML="This is my first paragraph.";
```

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <h1 id="myH1"></h1>
    <p id="myP"></p>

    <script>
      //document.getElementById("myH1").innerHTML="Welcome to my
Homepage";
      document.getElementById("myP").innerHTML="This is my first
paragraph.";
    </script>

    <p><strong>Note:</strong> The comment is not executed.</p>
  </body>
</html>
```

Result:

This is my first paragraph.

Note: The comment is not executed.

Usando Comentários para Prevenir Execuções – Parte 2

In the following example the comment is used to prevent the execution of a code block (can be suitable for debugging):

Example

```
/*
document.getElementById("myH1").innerHTML="Welcome to my Homepage";
document.getElementById("myP").innerHTML="This is my first paragraph.";
*/
```

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <h1 id="myH1"></h1>
    <p id="myP"></p>

    <script>
      /*
      document.getElementById("myH1").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>
```

Result:

Note: The comment-block is not executed.

Comentário de Fim de Linha

Using Comments at the End of a Line

In the following example the comment is placed at the end of a code line:

Example

```
var x=5;    // declare x and assign 5 to it
var y=x+2;  // declare y and assign x+2 to it
```

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p id="myP"></p>

    <script>
      var x=5;   // declare x and assign the value 5 to it
      var y=x+2; // declare y and assign the expression value x+2 to
it
      document.getElementById("myP").innerHTML=y // write the value
of y to myP
    </script>

    <p><strong>Note:</strong> The comments are not executed.</p>
  </body>
</html>
```

Result:

7

Note: The comments are not executed.

Variáveis

JavaScript variables are "containers" for storing information:

Example

```
var x=5;
var y=6;
var z=x+y;
```

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.**

Think of variables as containers for storing data.

Variáveis
Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var x=5;
      var y=6;
      var z=5+6;

      document.write(x + "<br>");
      document.write(y + "<br>");
      document.write(z + "<br>");
    </script>
  </body>
</html>
```

Result:

5
6
11

Regras para Variáveis

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 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)

Both JavaScript statements and JavaScript variables are case-sensitive.

Tipos de Dados

JavaScript Data Types

JavaScript variables can also hold other types of data, like text values (name="John Doe").

In JavaScript a text like "John Doe" is called a string.

There are many types of JavaScript variables, but for now, just think of numbers and strings.

When you assign a text value to a variable, put double or single quotes around the value.

When you assign a numeric value to a variable, do not put quotes around the value. If you put quotes around a numeric value, it will be treated as text.

Example

```
var pi=3.14;
var name="John Doe";
var answer='Yes I am!';
```

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var pi=3.14;
      var name="John Doe";
      var answer='Yes I am!';

      document.write(pi + "<br>");
      document.write(name + "<br>");
      document.write(answer + "<br>");
    </script>
  </body>
</html>
```

Result:

3.14
John Doe
Yes I am!

Declarando Variáveis

Declaring (Creating) JavaScript Variables

Creating a variable in JavaScript is most often referred to as "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";
```

However, 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, assigns the value "Volvo" to it, and put the value inside the HTML paragraph with id="demo":

Example

```
<p id="demo"></p>
var carname="Volvo";
document.getElementById("demo").innerHTML=carname;
```

It's a good programming practice to declare all the variables you will need, in one place, at the beginning of your code.

Declarando Várias Variáveis

One Statement, Many Variables

You can declare many variables in one statement. Just start the statement with **var** and separate the variables by comma:

```
var name="Doe", age=30, job="carpenter";
```

Your declaration can also span multiple lines:

```
var name="Doe",
age=30,
job="carpenter";
```

Valor Indefinido

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;
```

Redeclarando Variáveis

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;
```

Aritmética

JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:

Example

```
y=5;
x=y+2;
```

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p>Given that y=5, calculate x=y+2, and display the result.</p>
    <button onclick="myFunction()">Try it</button>

    <p id="demo"></p>

    <script>
      function myFunction(){
        var y=5;
        var x=y+2;
        var demoP=document.getElementById("demo")
        demoP.innerHTML="x=" + x;
      }
    </script>
  </body>
</html>
```

Result:

Given that y=5, calculate x=y+2, and display the result.

Try it

Result:

Given that y=5, calculate x=y+2, and display the result.

Try it

x=7

Após apertar o botão "*Try it*".

Tipo de Dados Dinâmico

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
```

String

JavaScript Strings

A string is a variable which 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"';
```

String
Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <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.write(carname1 + "<br>")
      document.write(carname2 + "<br>")
      document.write(answer1 + "<br>")
      document.write(answer2 + "<br>")
      document.write(answer3 + "<br>")
    </script>
  </body>
</html>
```

Result:

Volvo XC60
Volvo XC60
It's alright
He is called 'Johnny'
He is called "Johnny"

Números

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
```

Números
Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var x1=34.00;
      var x2=34;
      var y=123e5;
      var z=123e-5;

      document.write(x1 + "<br>")
      document.write(x2 + "<br>")
      document.write(y + "<br>")
      document.write(z + "<br>")
    </script>
  </body>
</html>
```

Result:

34
34
12300000
0.00123

Valor Booleano

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 more about conditional testing in a later chapter of this tutorial.

Array

JavaScript Arrays

The following code creates an Array called cars:

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

or (condensed array):

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

or (literal array):

Example

```
var cars=["Saab","Volvo","BMW"];
```

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 in later chapters of this tutorial.

Array
Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var i;
      var cars = new Array();
      cars[0] = "Saab";
      cars[1] = "Volvo";
      cars[2] = "BMW";

      for (i=0;i<cars.length;i++){
        document.write(cars[i] + "<br>");
      }
    </script>
  </body>
</html>
```

Result:

Saab
Volvo
BMW

Objetos

JavaScript Objects

An object is delimited by curly braces. Inside the braces the object's properties are defined as name and value pairs (name : value). The properties are separated by commas:

```
var person={firstname:"John", lastname:"Doe", id:5566};
```

The object (person) in the example above has 3 properties: firstname, lastname, and id.

Spaces and line breaks are not important. Your declaration can span multiple lines:

```
var person={
firstname : "John",
lastname  : "Doe",
id        :  5566
};
```

You can address the object properties in two ways:

Example

```
name=person.lastname;
name=person["lastname"];
```

Objetos
Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var person={
        firstname : "John",
        lastname  : "Doe",
        id        :  5566
      };
      document.write(person.lastname + "<br>");
      document.write(person["lastname"] + "<br>");
    </script>
  </body>
</html>
```

Result:

Doe
Doe

Valores: indefinido e nulo

Undefined and Null

Undefined is the value of a variable with no value.

Variables can be emptied by setting the value to **null**;

Example

```
cars=null;
person=null;
```

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var person;
      var car="Volvo";
      document.write(person + "<br>");
      document.write(car + "<br>");
      var car=null
      document.write(car + "<br>");
    </script>
  </body>
</html>
```

Result:

undefined
Volvo
null

Declarando Tipos de Variáveis

Declaring Variable Types

When you declare a new variable, you can declare its type using the "new" keyword:

```
var carname=new String;
var x=      new Number;
var y=      new Boolean;
var cars=   new Array;
var person= new Object;
```

JavaScript variables are all objects. When you declare a variable you create a new object.

Orientação a Objetos

- Em JavaScript, tudo é objeto:
  - uma *String*,
  - um *Array*,
  - uma data,
  - um número e
  - etc.

- Em JavaScript um objeto é um dado, com propriedades e métodos.

Propriedades e Métodos

Properties and Methods

Properties are **values** associated with an object.

Methods are **actions** that can be performed on objects.

A Real Life Object. A Car:

Properties:

car.name=Fiat

car.model=500

car.weight=850kg

car.color=white

Methods:

car.start()

car.drive()

car.brake()

The **properties** of the car include name, model, weight, color, etc.

All cars have these properties, but the values of those properties differ from car to car.

The **methods** of the car could be start(), drive(), brake(), etc.

All cars have these methods, but they are performed at different times.

Objetos em JavaScript

Objects in JavaScript:

In JavaScript, objects are data (variables), with properties and methods.

When you declare a JavaScript variable like this:

```
var txt = "Hello";
```

You actually create a JavaScript String object. The String object has a built-in property called length. For the string above, length has the value 5. The String object also have several built-in methods.

Properties:		Methods:
txt.length=5	**"Hello"**	txt.indexOf() txt.replace() txt.search()

In object oriented languages, properties and methods are often called **object members**.

Criando Objetos

Creating JavaScript Objects

Almost "everything" in JavaScript is an object. Strings, Dates, Arrays, Functions.

You can also create your own objects.

This example creates an object called "person", and adds four properties to it:

Example

```
person=new Object();
person.firstname="John";
person.lastname="Doe";
person.age=50;
person.eyecolor="blue";
```

There are many different ways to create new JavaScript objects, and you can also add properties and methods to existing objects.

Criando Objetos Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var person=new Object();
      person.firstname="John";
      person.lastname="Doe";
      person.age=50;
      person.eyecolor="blue";
      document.write(person.firstname + " is " + person.age + " years
old.");
    </script>
  </body>
</html>
```

Result:

John is 50 years old.

Acessando Propriedades

Accessing Object Properties

The syntax for accessing the property of an object is:

```
objectName.propertyName
```

This example uses the length property of the String object to find the length of a string:

```
var message="Hello World!";
var x=message.length;
```

The value of x, after execution of the code above will be:

```
12
```

Acessando Métodos

Accessing Object Methods

You can call a method with the following syntax:

```
objectName.methodName()
```

This example uses the toUpperCase() method of the String object, to convert a text to uppercase:

```
var message="Hello world!";
var x=message.toUpperCase();
```

The value of x, after execution of the code above will be:

```
HELLO WORLD!
```

Did You Know?

It is common in object oriented languages, to use camel-case function names.
You will more often see function names like someMethod() instead of some_method().

Funções

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>
```

Funções - Sintaxe

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.

Funções com Argumentos

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.

Funções com Argumentos Exemplo

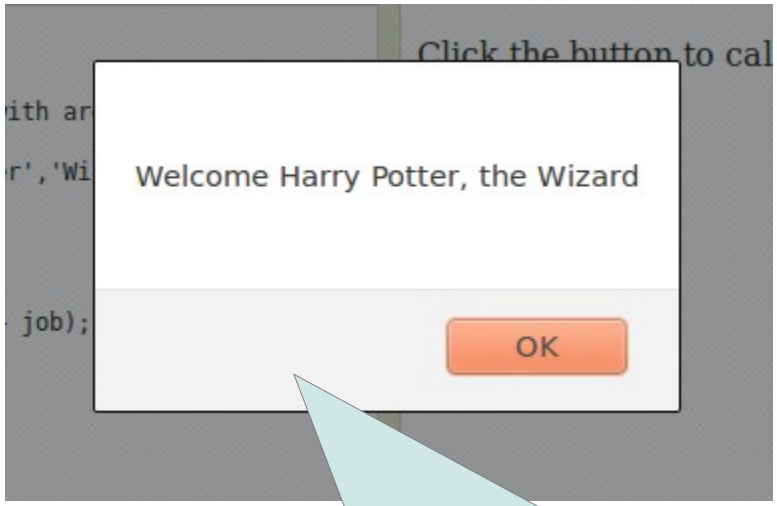
Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to call a function with arguments</p>

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

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

Result:

Click the button to call a function with arguments

Try it



Mensagem após botão "*Try it*" ser pressionado.

Funções com Argumentos
Outro Exemplo

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

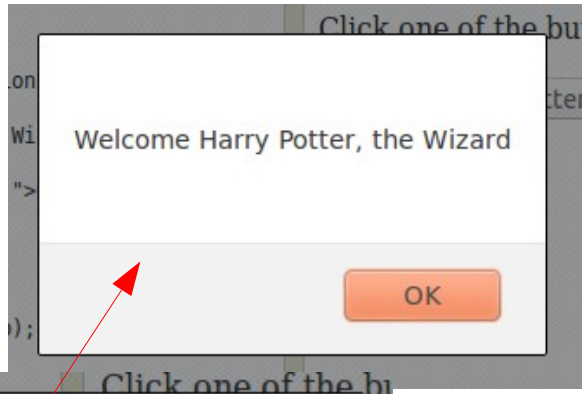
Example

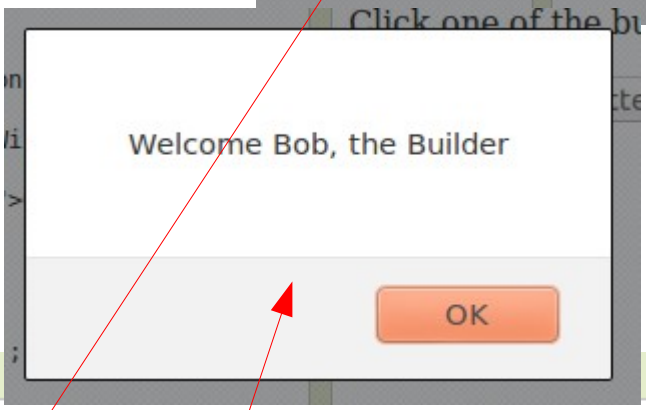
```
<button onclick="myFunction('Harry Potter','Wizard')">Try it</button>
<button onclick="myFunction('Bob','Builder')">Try it</button>
```

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

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click one of the buttons to call a function with arguments</p>

    <button onclick="myFunction('Harry Potter','Wizard')">Click for
Harry Potter</button>
    <button onclick="myFunction('Bob','Builder')">Click for
Bob</button>

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





Result:

Click one of the buttons to call a function with arguments

Click for Harry Potter | Click for Bob

Funções com Valor de Retorno

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 returnvalue:

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

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

Funções com Valor de Retorno

You can also use the returnvalue 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 returnvalue 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
```

Funções com Valor de Retorno

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.

Funções com Valor de Retorno Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p>This example calls a function which perfoms 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>
```

Result:

This example calls a function which perfoms a calculation, and returns the result:

12

Variáveis Locais, Globais e Tempo de Vida de Cada Qual

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.

Global JavaScript Variables

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

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.

Atenção!

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.

Operadores

= is used to assign values.

+ is used to add 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;
z=2;
x=y+z;
```

The result of *x* will be:

```
7
```

Operadores
Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to calculate 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>
```

Result:

Click the button to calculate x.

Try it

Result:

Click the button to calculate x.

Try it

7

Após clique no botão “*Try it*”.

Operadores Aritméticos

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 of x	Result of y
+	Addition	x=y+2	7	5
-	Subtraction	x=y-2	3	5
*	Multiplication	x=y*2	10	5
/	Division	x=y/2	2.5	5
%	Modulus (division remainder)	x=y%2	1	5
++	Increment	x=++y	6	6
		x=y++	5	6
--	Decrement	x=--y	4	4
		x=y--	5	4

Operadores de Atribuição

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
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

O Operador "+" Usado com *Strings*

The + operator can also be used to add string variables or text values together.

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
```

Source Code: Submit Co

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to create and add string variables.</p>
    <button onclick="myFunction()">Try it</button>

    <p id="demo"></p>

    <script>
      function myFunction(){
        txt1="What a very";
        txt2="nice day";
        txt3=txt1+txt2;
        document.getElementById("demo").innerHTML=txt3;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to create and add string variables.

Try it

Result:

Click the button to create and add string variables.

Try it

What a verynice day

Adicionando Espaço em Branco Exemplo 1

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
```

Source Code: Submit Co

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to create and add string variables.</p>
    <button onclick="myFunction()">Try it</button>

    <p id="demo"></p>

    <script>
      function myFunction(){
        txt1="What a very ";
        txt2="nice day";
        txt3=txt1+txt2;
        document.getElementById("demo").innerHTML=txt3;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to create and add string variables.

Try it

Result:

Click the button to create and add string variables.

Try it

What a very nice day

Adicionando Espaço em Branco Exemplo 2

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
```

Source Code: Submit Co

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to create and add string variables.</p>
    <button onclick="myFunction()">Try it</button>

    <p id="demo"></p>

    <script>
      function myFunction(){
        txt1="What a very ";
        txt2="nice day";
        txt3=txt1+" "+txt2;
        document.getElementById("demo").innerHTML=txt3;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to create and add string variables.

Try it

Result:

Click the button to create and add string variables.

Try it

What a very nice day

Adicionando *Strings* e Números

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
```

Adicionando *Strings* e Números Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to add numbers and strings.</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;

        var demoP=document.getElementById("demo");
        demoP.innerHTML=x + "<br>" + y + "<br>" + z;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to add numbers and strings.

Try it

Result:

Click the button to add numbers and strings.

Try it

10
55
Hello5

Operadores de Comparação

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 variable

Given that **x=5**, the table below explains the comparison operators:

Operator	Description	Comparing	Returns
==	is equal to	x==8	*false*
		x==5	*true*
===	is exactly equal to (value and type)	x==="5"	*false*
		x===5	*true*
!=	is not equal	x!=8	*true*
!==	is not equal (neither value nor type)	x!=="5"	*true*
		x!==5	*false*
>	is greater than	x>8	*false*
<	is less than	x<8	*true*
>=	is greater than or equal to	x>=8	*false*
<=	is less than or equal to	x<=8	*true*

Como Usar?

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) x="Too young";
```

Operadores Lógicos

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

Example

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":

```
voteable=(age<18)?"Too young":"Old enough";
```

Operadores Lógicos Exemplo

Source Code: Submi

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to check the age.</p>

    Age:<input id="age" value="18" />
    <p>Old enough to vote?</p>
    <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;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to check the age.

Age: 18

Old enough to vote?

Try it

Result:

Click the button to check the age.

Age: 18

Old enough to vote?

Try it

Old enough

Estruturas Condicionais

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:

- **if statement** - use this statement to execute some code only if a specified condition is true
- **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false
- **if...else if....else statement** - use this statement to select one of many blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

if

If Statement

Use the if statement to execute some code only if a specified condition is true.

Syntax

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

Note that *if* is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

Example

Make a "Good day" greeting if the time if less than 20:00:

```
if (time<20)
  {
  x="Good day";
  }
```

The result of *x* will be:

```
Good day
```

Notice that there is no ..else.. in this syntax. You tell the browser to execute some code **only if the specified condition is true**.

if

Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to get a "Good day" greeting if the time is
less than 20:00.</p>
    <button onclick="myFunction()">Try it</button>

    <p id="demo"></p>

    <script>
      function myFunction(){
        var x="";
        var time=new Date().getHours();
        if (time<20) {
          x="Good day";
        }
        document.getElementById("demo").innerHTML=x;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to get a "Good day" greeting if the time is less than 20:00.

Try it

Result:

Click the button to get a "Good day" greeting if the time is less than 20:00.

Try it

Good day

if ... else

If...else Statement

Use the if....else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax

```
if (condition)
  {
  code to be executed if condition is true
  }
else
  {
  code to be executed if condition is not true
  }
```

Example

If the time is less than 20:00, you will get a "Good day" greeting, otherwise you will get a "Good evening" greeting

```
if (time<20)
  {
  x="Good day";
  }
else
  {
  x="Good evening";
  }
```

The result of *x* will be:

```
Good day
```

if ... else
Exemplo

Source Code: Subn

```
<!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 x="";
        var time=new Date().getHours();
        if (time<20){
          x="Good day";
        }
        else{
          x="Good evening";
        }
        document.getElementById("demo").innerHTML=x;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to get a time-based greeting.

Try it

Result:

Click the button to get a time-based greeting.

Try it

Good day

if ... else ... if ... else

If...else if...else Statement

Use the if....else if...else statement to select one of several blocks of code to be executed.

Syntax

```
if (condition1)
  {
  code to be executed if condition1 is true
  }
else if (condition2)
  {
  code to be executed if condition2 is true
  }
else
  {
  code to be executed if neither condition1 nor condition2 is true
  }
```

if ... else ... if ... else

Example

If the time is less than 10:00, you will get a "Good morning" greeting, if not, but the time is less than 20:00, you will get a "Good day" greeting, otherwise you will get a "Good evening" greeting:

```
if (time<10)
  {
  x="Good morning";
  }
else if (time<20)
  {
  x="Good day";
  }
else
  {
  x="Good evening";
  }
```

The result of *x* will be:

```
Good day
```

if ... else ... if ... else Exemplo

Source Code: Subn

```
<!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 x="";
        var time=new Date().getHours();
        if (time<10){
          x="Good morning";
        }
        else
          if (time<20){
            x="Good day";
          }
          else{
            x="Good evening";
          }
          document.getElementById("demo").innerHTML=x;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to get a time-based greeting.

Try it

Result:

Click the button to get a time-based greeting.

Try it

Good day

switch ... case

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(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```

This is how it works: First we have a single expression *n* (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. Use **break** to prevent the code from running into the next case automatically.

switch ... case

Example

Display today's weekday-name. Note that Sunday=0, Monday=1, Tuesday=2, etc:

```
var day=new Date().getDay();
switch (day)
{
case 0:
  x="Today it's Sunday";
  break;
case 1:
  x="Today it's Monday";
  break;
case 2:
  x="Today it's Tuesday";
  break;
case 3:
  x="Today it's Wednesday";
  break;
case 4:
  x="Today it's Thursday";
  break;
case 5:
  x="Today it's Friday";
  break;
case 6:
  x="Today it's Saturday";
  break;
}
```

The result of *x* will be:

```
Today it's Saturday
```

switch ... *case* Exemplo

Source Code: Submit

```
<!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 x;
        var d=new Date().getDay();
        switch (d){
          case 0:
            x="Today it's Sunday";
            break;
          case 1:
            x="Today it's Monday";
            break;
          case 2:
            x="Today it's Tuesday";
            break;
          case 3:
            x="Today it's Wednesday";
            break;
          case 4:
            x="Today it's Thursday";
            break;
          case 5:
            x="Today it's Friday";
            break;
          case 6:
            x="Today it's Saturday";
            break;
        }
        document.getElementById("demo").innerHTML=x;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to display what day it is today.

Try it

Result:

Click the button to display what day it is today.

Try it

Today it's Saturday

switch ... *case* com *default*

The *default* Keyword

Use the *default* keyword to specify what to do if there is no match:

Example

If it is NOT Saturday or Sunday, then write a default message:

```
var day=new Date().getDay();
switch (day)
{
case 6:
  x="Today it's Saturday";
  break;
case 0:
  x="Today it's Sunday";
  break;
default:
  x="Looking forward to the Weekend";
}
```

The result of *x* will be:

```
Today it's Saturday
```

switch ... *case* com *default* Exemplo

Source Code: Submit Code »

```
<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 x;
        var d=new Date().getDay();
        switch (d){
          case 1:
            x="Today it's Monday";
            break;
          case 2:
            x="Today it's Tuesday";
            break;
          default:
            x="Today isn't Monday or Tuesday.";
        }
        document.getElementById("demo").innerHTML=x;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to display a message based on what day it is today.

Try it

Result:

Click the button to display a message based on what day it is today.

Try it

Today isn't Monday or Tuesday.

Estruturas de Repetição

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:

```
document.write(cars[0] + "<br>");
document.write(cars[1] + "<br>");
document.write(cars[2] + "<br>");
document.write(cars[3] + "<br>");
document.write(cars[4] + "<br>");
document.write(cars[5] + "<br>");
```

You can write:

```
for (var i=0;i<cars.length;i++)
{
document.write(cars[i] + "<br>");
}
```

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

Estruturas de Repetição Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      cars=["BMW","Volvo","Saab","Ford"];
      for(var i=0;i<cars.length;i++){
        document.write(cars[i] + "<br>");
      }
    </script>
  </body>
</html>
```

Result:

BMW
Volvo
Saab
Ford

for

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)
  {
  the 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 (var i=0; i<5; i++)
  {
  x=x + "The number is " + i + "<br>";
  }
```

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.

for
Exemplo

Source Code: Submit

```
<!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 x="";
        for(var i=0;i<5;i++){
          x=x + "The number is " + i + "<br>";
        }
        document.getElementById("demo").innerHTML=x;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to loop through a block of code five times.

Try it

Result:

Click the button to loop through a block of code five times.

Try it

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4

for ... in

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};

for (x in person)
  {
  txt=txt + person[x];
  }
```

for ... in

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to loop through the properties of an object
named "person".</p>
    <button onclick="myFunction()">Try it</button>
    <p id="demo"></p>

    <script>
      function myFunction(){
        var x;
        var txt="";
        var person={fname:"John",lname:"Doe",age:25};

        for(x in person){
          txt=txt + person[x];
        }

        document.getElementById("demo").innerHTML=txt;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to loop through the properties of an object named "person".

Try it

Result:

Click the button to loop through the properties of an object named "person".

Try it

JohnDoe25

while

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

The loop in this example will continue to run as long as the variable i is less than 5:

Example

```
while (i<5)
  {
  x=x + "The number is " + i + "<br>";
  i++;
  }
```

If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

while
Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to loop through a block of as long as
<em>i</em> is less than 5.</p>
    <button onclick="myFunction()">Try it</button>
    <p id="demo"></p>

    <script>
      function myFunction(){
        var x="",i=0;
        while (i<5){
          x=x + "The number is " + i + "<br>";
          i++;
        }
        document.getElementById("demo").innerHTML=x;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to loop through a block of as long as *i* is less than 5.

Try it

Result:

Click the button to loop through a block of as long as *i* is less than 5.

Try it

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4

do ... while

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
  {
  x=x + "The number is " + i + "<br>";
  i++;
  }
while (i<5);
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

do ... while
Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to loop through a block of as long as
<em>i</em> is less than 5.</p>
    <button onclick="myFunction()">Try it</button>
    <p id="demo"></p>

    <script>
      function myFunction(){
        var x="",i=0;
        do{
          x=x + "The number is " + i + "<br>";
          i++;
        }while(i<5)
        document.getElementById("demo").innerHTML=x;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to loop through a block of as long as *i* is less than 5.

Try it

Result:

Click the button to loop through a block of as long as *i* is less than 5.

Try it

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4

break e *continue*

The break statement "jumps out" of a loop.

The continue statement "jumps over" one iteration in the loop.

break

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;
    }
  x=x + "The number is " + i + "<br>";
  }
```

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;
  x=x + "The number is " + i + "<br>";
  }
```

break
Exemplo

Source Code:

```
<!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 x="",i=0;
        for (i=0;i<10;i++){
          if (i==3){
            break;
          }
          x=x + "The number is " + i + "<br>";
        }
        document.getElementById("demo").innerHTML=x;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to do a loop with a break.

Try it

Result:

Click the button to do a loop with a break.

Try it

The number is 0
The number is 1
The number is 2

continue

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;
  x=x + "The number is " + i + "<br>";
  }
```

continue

Exemplo

Source Code:

```
<!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 x="",i=0;
        for (i=0;i<10;i++){
          if (i==3){
            continue;
          }
          x=x + "The number is " + i + "<br>";
        }
        document.getElementById("demo").innerHTML=x;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to do a loop which will skip the step where i=3.

Try it

Result:

Click the button to do a loop which will skip the step where i=3.

Try it

The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9

throw e try catch

The **try** statement lets you to test a block of code for errors.

The **catch** statement lets you handle the error.

The **throw** statement lets you create custom errors.

Errors Will Happen!

When the JavaScript engine is executing JavaScript code, different errors can occur:

It can be syntax errors, typically coding errors or typos made by the programmer.

It can be misspelled or missing features in the language (maybe due to browser differences).

It can be errors due to wrong input, from a user, or from an Internet server.

And, of course, it can be many other unforeseeable things.

JavaScript Throws Errors

When an error occurs, when something goes wrong, the JavaScript engine will normally stop, and generate an error message.

The technical term for this is: JavaScript will **throw** an error.

try ... catch

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.

Syntax

```
try
  {
  //Run some code here
  }
catch(err)
  {
  //Handle errors here
  }
```

try ... catch
Exemplo 1

Examples

In the example below we have deliberately made a typo in the code in the try block.

The catch block catches the error in the try block, and executes code to handle it:

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
var txt="";
function message()
{
try
  {
  adddlert("Welcome guest!");
  }
catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
  }
}
</script>
</head>

<body>
<input type="button" value="View message" onclick="message()">
</body>

</html>
```

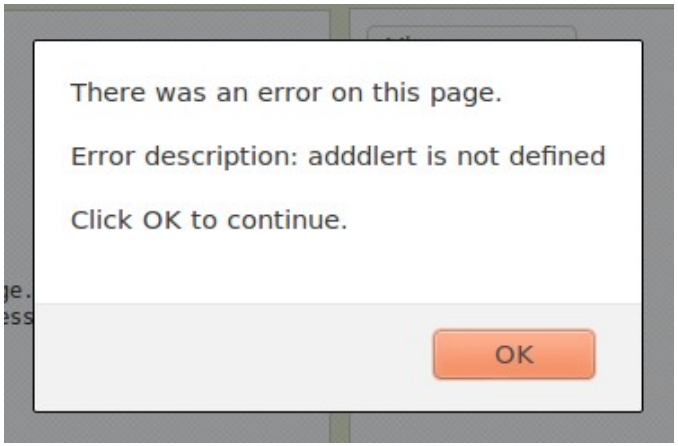
try ... catch
Exemplo 2

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      var txt="";
      function message(){
        try{
          adddlert("Welcome guest!");
        }
        catch(err){
          txt="There was an error on this page.\n\n";
          txt+="Error description: " + err.message + "\n\n";
          txt+="Click OK to continue.\n\n";
          alert(txt);
        }
      }
    </script>
  </head>
  <body>
    <input type="button" value="View message" onclick="message()" />
  </body>
</html>
```

Result:

View message



throw

The Throw Statement

The throw statement allows you to create a custom error.

The correct technical term is to create or **throw an exception**.

If you use the throw statement together with try and catch, you can control program flow and generate custom error messages.

Syntax

```
throw exception
```

The exception can be a JavaScript String, a Number, a Boolean or an Object.

throw

Exemplo 1

Example

This example examines the value of an input variable. If the value is wrong, an exception (error) is thrown. The error is caught by the catch statement and a custom error message is displayed:

Example

```
<script>
function myFunction()
{
try
  {
  var x=document.getElementById("demo").value;
  if(x=="")    throw "empty";
  if(isNaN(x)) throw "not a number";
  if(x>10)     throw "to high";
  if(x<5)      throw "too low";
  }
catch(err)
  {
  var y=document.getElementById("mess");
  y.innerHTML="Error: " + err + ".";
  }
}
</script>

<h1>My First JavaScript</h1>
<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="mess"></p>
```

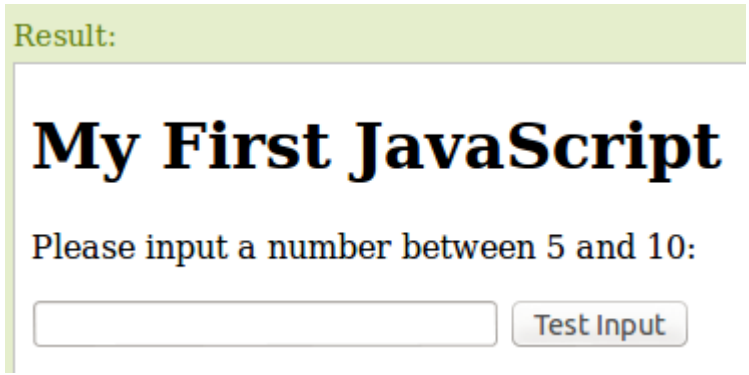
Note that the example above will also throw an error if the getElementById function fails.

throw
Exemplo 2

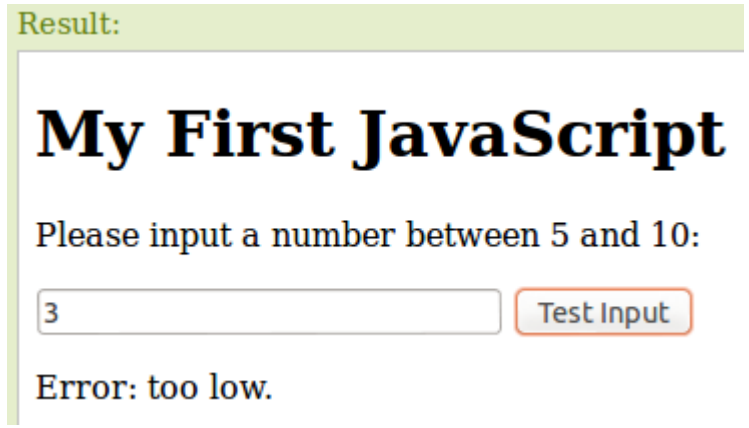
Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      function myFunction(){
        try{
          var x=document.getElementById("demo").value;
          if(x=="")    throw "empty";
          if(isNaN(x)) throw "not a number";
          if(x>10)     throw "to high";
          if(x<5)      throw "too low";
        }
        catch(err){
          var y=document.getElementById("mess");
          y.innerHTML="Error: " + err + ".";
        }
      }
    </script>

    <h1>My First JavaScript</h1>
    <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="mess"></p>
  </body>
</html>
```

Result:



Result:



Validação de Formulário

JavaScript Form Validation

JavaScript can be used to validate data in HTML forms before sending off the content to a server.

Form data that typically are checked by a JavaScript could be:

- has the user left required fields empty?
- has the user entered a valid e-mail address?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Required Fields

The function below checks if a field has been left empty. If the field is blank, an alert box alerts a message, the function returns false, and the form will not be submitted:

```
function validateForm()
{
var x=document.forms["myForm"]["fname"].value;
if (x==null || x=="")
  {
  alert("First name must be filled out");
  return false;
  }
}
```

Validação de Formulário Exemplo

The function in prior slide could be called when a form is submitted:

Example

```
<form name="myForm" action="demo_form.asp" onsubmit="return validateForm()"
method="post">
First name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

Source Code:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function validateForm(){
        var x=document.forms["myForm"]["fname"].value;
        if (x==null || x==""){
          alert("First name must be filled out");
          return false;
        }
      }
    </script>
  </head>
  <body>
    <form name="myForm" action="demo_form.asp" onsubmit="return
validateForm()" method="post">
      First name: <input type="text" name="fname">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Result:

First name: Submit

Validação de *E-mail*

E-mail Validation

The function below checks if the content has the general syntax of an email.

This means that the input data must contain an @ sign and at least one dot (.). Also, the @ must not be the first character of the email address, and the last dot must be present after the @ sign, and minimum 2 characters before the end:

```
function validateForm()
{
var x=document.forms["myForm"]["email"].value;
var atpos=x.indexOf("@");
var dotpos=x.lastIndexOf(".");
if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)
  {
  alert("Not a valid e-mail address");
  return false;
  }
}
```

The function above could be called when a form is submitted:

Example

```
<form name="myForm" action="demo_form.asp" onsubmit="return
validateForm();" method="post">
Email: <input type="text" name="email">
<input type="submit" value="Submit">
</form>
```

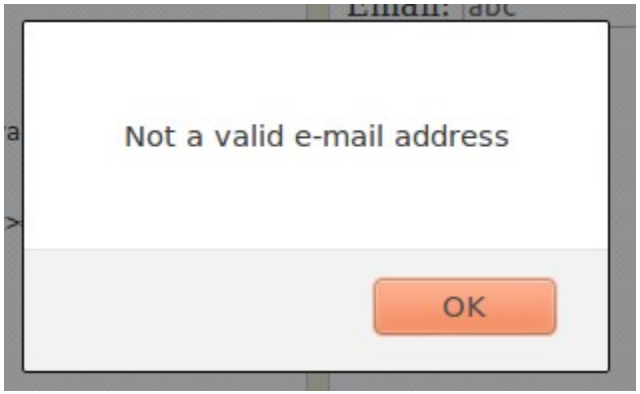
Validação de *E-mail* Exemplo

Source Code: Submit Co

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function validateForm(){
        var x=document.forms["myForm"]["email"].value;
        var atpos=x.indexOf("@");
        var dotpos=x.lastIndexOf(".");
        if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length){
          alert("Not a valid e-mail address");
          return false;
        }
      }
    </script>
  </head>
  <body>
    <form name="myForm" action="demo_form.asp" onsubmit="return
validateForm();" method="post">
      Email: <input type="text" name="email">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Result:

Email: abc Submit



Windows 8

JavaScript in Windows 8

Microsoft supports JavaScript for creating Windows 8 apps.
JavaScript is definitely the future for both the Internet and Windows.

JS HTML DOM

HTML DOM (*Document Object Model*)

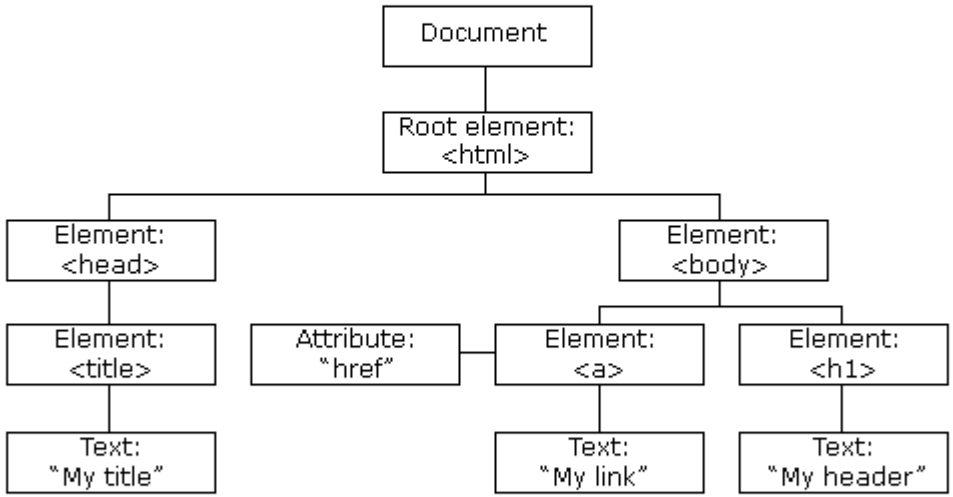
With the HTML DOM, JavaScript can access all the elements of an HTML document.

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree



With a programmable object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can react to all the events in the page

Encontrando Elementos HTML pelo *id*

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are a couple of ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name

Finding HTML Elements by Id

The easiest way to find HTML elements in the DOM, is by using the element id.

This example finds the element with id="intro":

Example

```
var x=document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in x).

If the element is not found, x will contain null.

Encontrando Elementos HTML pelo *id* Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p id="intro">Hello World!</p>
    <p>This example demonstrates the <b>getElementById</b> method!</p>

    <script>
      x=document.getElementById("intro");
      document.write("<p>The text from the intro paragraph: " +
x.innerHTML + "</p>");
    </script>
  </body>
</html>
```

Result:

Hello World!

This example demonstrates the **getElementById** method!

The text from the intro paragraph: Hello World!

Encontrando Elementos HTML pela *tag*

Finding HTML Elements by Tag Name

This example finds the element with id="main", and then finds all <p> elements inside "main":

Example

```
var x=document.getElementById("main");
var y=x.getElementsByTagName("p");
```

Finding elements by class name does not work in Internet Explorer 5,6,7, and 8.

Encontrando Elementos HTML pela *tag* Exemplo

Source Code: Submit Code

```
<!DOCTYPE html>
<html>
  <body>
    <p>Hello World!</p>

    <div id="main">
      <p>The DOM is very useful.</p>
      <p>This example demonstrates the <b>getElementsByTagName</b>
method</p>
    </div>

    <script>
      var x=document.getElementById("main");
      var y=x.getElementsByTagName("p");
      document.write('First paragraph inside "main" is ' +
y[0].innerHTML);
    </script>
  </body>
</html>
```

Result:

Hello World!

The DOM is very useful.

This example demonstrates the **getElementsByTagName** method

First paragraph inside "main" is The DOM is very useful.

Alterando o Fluxo de Saída HTML

The HTML DOM allows JavaScript to change the content of HTML elements.

Changing the HTML Output Stream

JavaScript can create dynamic HTML content:

Date: Mon Jan 21 2013 11:34:37 GMT-0200 (BRST)

In JavaScript, document.write() can be used to write directly to the HTML output stream:

Example

```
<!DOCTYPE html>
<html>
<body>

<script>
document.write(Date());
</script>

</body>
</html>
```

Never use document.write() after the document is loaded. It will overwrite the document.

Alterando o Fluxo de Saída HTML Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      document.write(Date());
    </script>
  </body>
</html>
```

Result:

Mon Jan 21 2013 11:36:58 GMT-0200 (BRST)

Alterando o Conteúdo HTML Parte 01

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the **innerHTML** property.

To change the content of an HTML element, use this syntax:

```
document.getElementById(id).innerHTML=new HTML
```

This example changes the content of a <p> element:

Example

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML="New text!";
</script>

</body>
</html>
```

Alterando o Conteúdo HTML Parte 01 - Exemplo

Source Code: Subm

```
<!DOCTYPE html>
<html>
  <body>

    <p id="p1">Hello World!</p>

    <script>
      document.getElementById("p1").innerHTML="New text!";
    </script>

    <p>The paragraph above was changed by a script.</p>

  </body>
</html>
```

Result:

New text!

The paragraph above was changed by a script.

Alterando o Conteúdo HTML Parte 02

This example changes the content of an <h1> element:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 id="header">Old Header</h1>

<script>
var element=document.getElementById("header");
element.innerHTML="New Header";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains an <h1> element with id="header"
- We use the HTML DOM to get the element with id="header"
- A JavaScript changes the content (innerHTML) of that element

Alterando o Conteúdo HTML Parte 02 - Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>

    <h1 id="header">Old Header</h1>

    <script>
      var element=document.getElementById("header");
      element.innerHTML="New Header";
    </script>

    <p>"Old Header" was changed to "New Header"</p>

  </body>
</html>
```

Result:

New Header

"Old Header" was changed to "New Header"

Alterando um Atributo HTML

Changing an HTML Attribute

To change the attribute of an HTML element, use this syntax:

```
document.getElementById(id).attribute=new value
```

This example changes the src attribute of an <img> element:

Example

```
<!DOCTYPE html>
<html>
<body>

<img id="image" src="smiley.gif">

<script>
document.getElementById("image").src="landscape.jpg";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains an <img> element with id="image"
- We use the HTML DOM to get the element with id="image"
- A JavaScript changes the src attribute of that element from "smiley.gif" to "landscape.jpg"

Alterando um Atributo HTML Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>

    <img id="image" src="smiley.gif" width="160" height="120">

    <script>
      document.getElementById("image").src="landscape.jpg";
    </script>

    <p>The original image was smiley.gif, but the script changed it
to landscape.jpg</p>

  </body>
</html>
```

Result:

The original image was smiley.gif, but the script changed it to landscape.jpg

Alterando um Estilo HTML Parte 01

The HTML DOM allows JavaScript to change the style of HTML elements.

Changing HTML Style

To change the style of an HTML element, use this syntax:

```
document.getElementById(id).style.property=new style
```

The following example changes the style of a <p> element:

Example

```
<html>
<body>

<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color="blue";
</script>

<p>The paragraph above was changed by a script.</p>

</body>
</html>
```

Alterando um Estilo HTML Parte 01 - Exemplo

Source Code: Submit

```
<!DOCTYPE html>
<html>
  <body>

    <p id="p1">Hello World!</p>
    <p id="p2">Hello World!</p>

    <script>
      document.getElementById("p2").style.color="blue";
      document.getElementById("p2").style.fontFamily="Arial";
      document.getElementById("p2").style.fontSize="larger";
    </script>

    <p>The paragraph above was changed by a script.</p>

  </body>
</html>
```

Result:

Hello World!

Hello World!

The paragraph above was changed by a script.

Alterando um Estilo HTML Parte 02

This example changes the style of the HTML element with id="id1", when the user clicks a button:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id1">My Heading 1</h1>
<button type="button"
onclick="document.getElementById('id1').style.color='red'">
Click Me!</button>

</body>
</html>
```

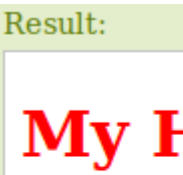
Alterando um Estilo HTML Parte 02 - Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>

    <h1 id="id1">My Heading 1</h1>
    <button type="button"
onclick="document.getElementById('id1').style.color='red'"> Click
Me!</button>

  </body>
</html>
```

Result:

Result:



Alterando um Estilo HTML Parte 03

- Como fazer um elemento HTML ficar invisível e também visível;

- Direto ao exemplo no próximo *slide*:

  Observação: existem diversas propriedades de estilo que podem ser trabalhadas, para acessar a lista completa clique aqui.

Alterando um Estilo HTML Parte 03 - Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>

    <p id="p1">This is a text. This is a text. This is a text. This
is a text. This is a text. This is a text. This is a text.</p>

    <input type="button" value="Hide text"
onclick="document.getElementById('p1').style.visibility='hidden'" />
    <input type="button" value="Show text"
onclick="document.getElementById('p1').style.visibility='visible'" />

  </body>
</html>
```

Result:

This is a text. This is a text. This is a text. This is a text. This is a text. This is a text. This is a text.

Hide text Show text

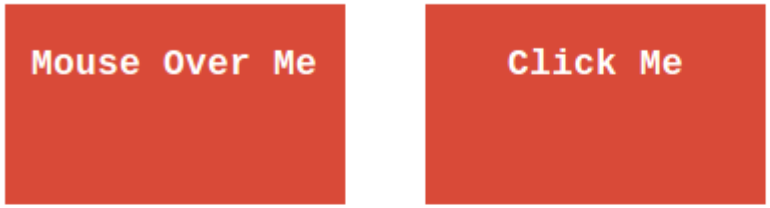
Result:

Hide text Show text

Eventos

HTML DOM allows JavaScript to react to HTML events.

Example



Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

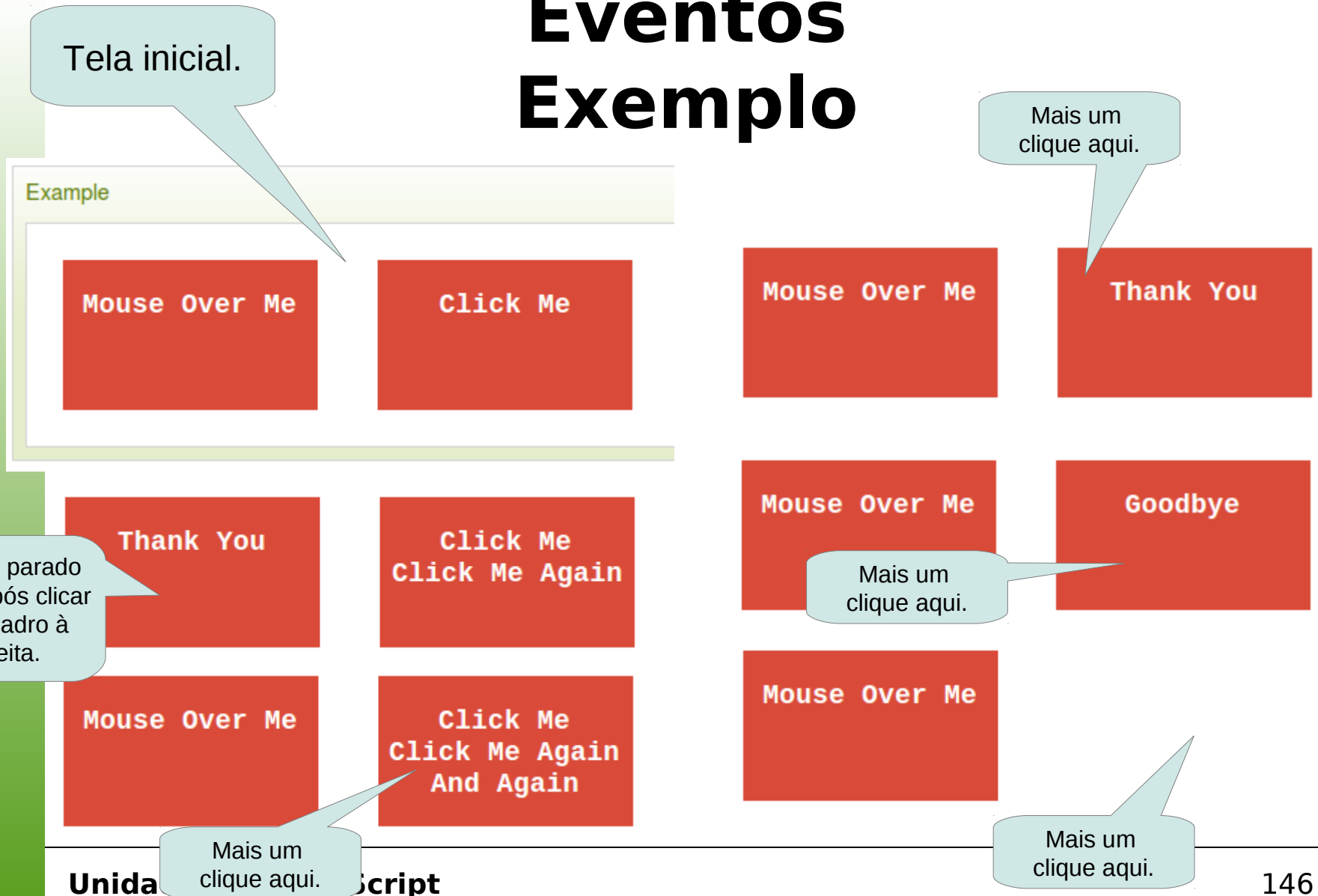
To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

```
onclick=JavaScript
```

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

Eventos Exemplo



Exemplo 2

In this example, the content of the <h1> element is changed when a user clicks on it:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>
</body>
</html>
```

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>

    <h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>

  </body>
</html>
```

Result:

Click on this text!

Result:

Ooops!

Exemplo 3

In this example, a function is called from the event handler:

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function changetext(id)
{
id.innerHTML="Ooops!";
}
</script>
</head>
<body>
<h1 onclick="changetext(this)">Click on this text!</h1>
</body>
</html>
```

Source Code: Submit

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function changetext(id){
        id.innerHTML="Ooops!";
      }
    </script>
  </head>
  <body>
    <h1 onclick="changetext(this)">Click on this text!</h1>
  </body>
</html>
```

Result:

Click on this text!

Result:

Ooops!

Associação de Eventos a Elementos HTML

HTML Event Attributes

To assign events to HTML elements you can use event attributes.

Example

Assign an onclick event to a button element:

```
<button onclick="displayDate()">Try it</button>
```

In the example above, a function named *displayDate* will be executed when the button is clicked.

Source Code: Submit C

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to execute the <em>displayDate()</em>
function.</p>
    <button onclick="displayDate()">Try it</button>

    <script>
      function displayDate(){
        document.getElementById("demo").innerHTML=Date();
      }
    </script>

    <p id="demo"></p>
  </body>
</html>
```

Result:

Click the button to execute the *displayDate()* function.

Try it

Result:

Click the button to execute the *displayDate()* function.

Try it

Mon Jan 21 2013 16:07:03 GMT-0200 (BRST)

Associação de Eventos Usando HTML DOM

Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript:

Example

Assign an onclick event to a button element:

```
<script>
document.getElementById("myBtn").onclick=function(){displayDate()};
</script>
```

In the example above, a function named *displayDate* is assigned to an HTML element with the id=myButn".

The function will be executed when the button is clicked.

Associação de Eventos Usando HTML DOM Exemplo

Source Code: Submit C

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to execute the <em>displayDate()</em>
function.</p>
    <button id="myBtn">Try it</button>

    <script>
      document.getElementById("myBtn").onclick=function()
{displayDate()};

      function displayDate(){
        document.getElementById("demo").innerHTML=Date();
      }
    </script>

    <p id="demo"></p>
  </body>
</html>
```

Result:

Click the button to execute the *displayDate()* function.

Try it

Result:

Click the button to execute the *displayDate()* function.

Try it

Mon Jan 21 2013 16:12:13 GMT-0200 (BRST)

onload / onunload

The onload and onunload Events

The onload and onunload events are triggered when the user enters or leaves the page.

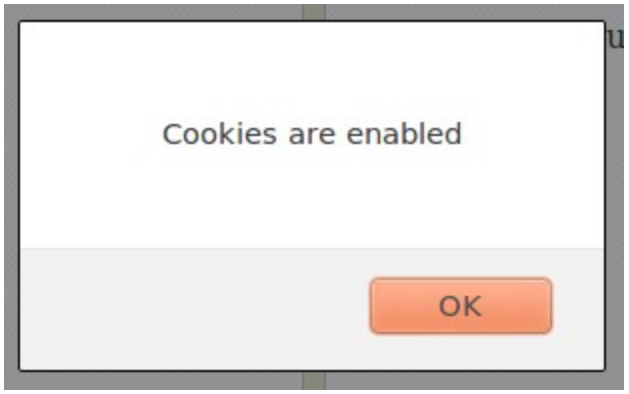
The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The onload and onunload events can be used to deal with cookies.

Example

```
<body onload="checkCookies()">
```

onload / onunload
Exemplo



Source Code:

```
<!DOCTYPE html>
<html>
  <body onload="checkCookies()">
    <script>
      function checkCookies(){
        if (navigator.cookieEnabled==true){
          alert("Cookies are enabled")
        }
        else{
          alert("Cookies are not enabled")
        }
      }
    </script>

    <p>An alert box should tell you if your browser has enabled
cookies or not.</p>
  </body>
</html>
```

Result:

An alert box should tell you if your browser has enabled cookies or not.

onchange

The onchange Event

The onchange event are often used in combination with validation of input fields.

Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.

Example

```
<input type="text" id="fname" onchange="upperCase()">
```

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function myFunction(){
        var x=document.getElementById("fname");
        x.value=x.value.toUpperCase();
      }
    </script>
  </head>
  <body>
    Enter your name: <input type="text" id="fname"
onchange="myFunction()">
    <p>When you leave the input field, a function is triggered which
transforms the input text to upper case.</p>
  </body>
</html>
```

Result:

Enter your name:

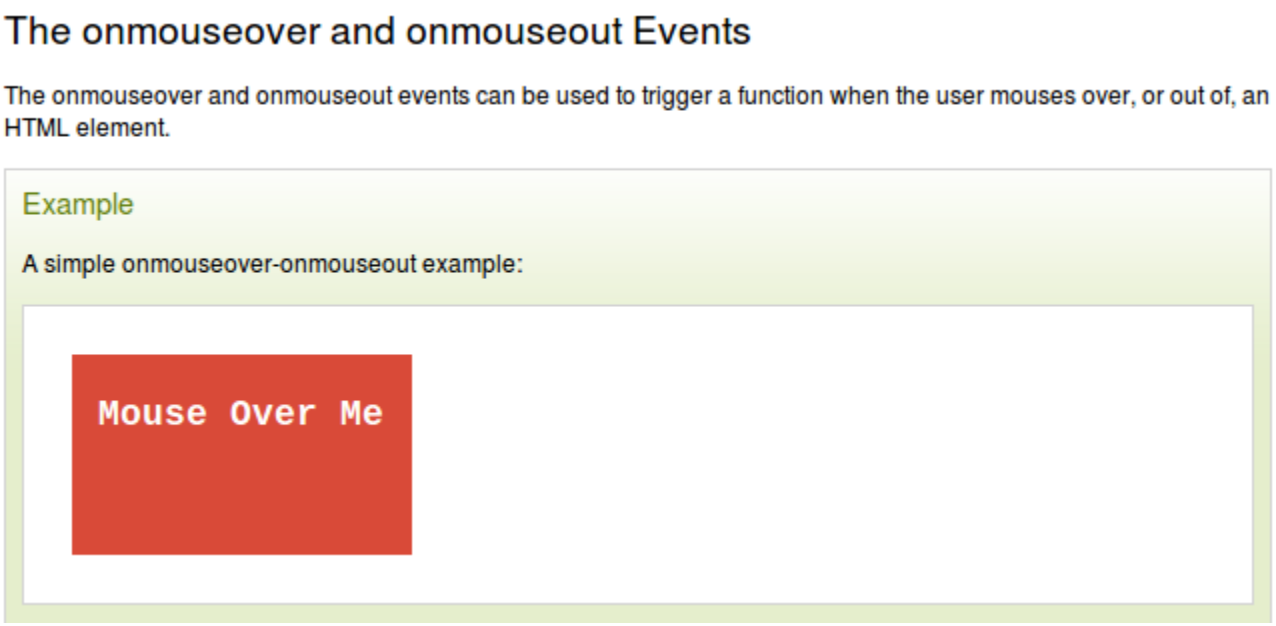
When you leave the input field, a function is triggered which transforms the input text to upper case.

Result:

Enter your name: EDWAR SALIBA

When you leave the input field, a function is triggered which transforms the input text to upper case.

onmouseover / onmouseout



onmouseover / onmouseout Exemplo

Source Code: Subm

```
<!DOCTYPE html>
<html>
  <body>
    <div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;
padding:40px;">Mouse Over Me</div>

    <script>
      function mOver(obj){
        obj.innerHTML="Thank You"
      }

      function mOut(obj){
        obj.innerHTML="Mouse Over Me"
      }
    </script>
  </body>
</html>
```

Result:

Result:

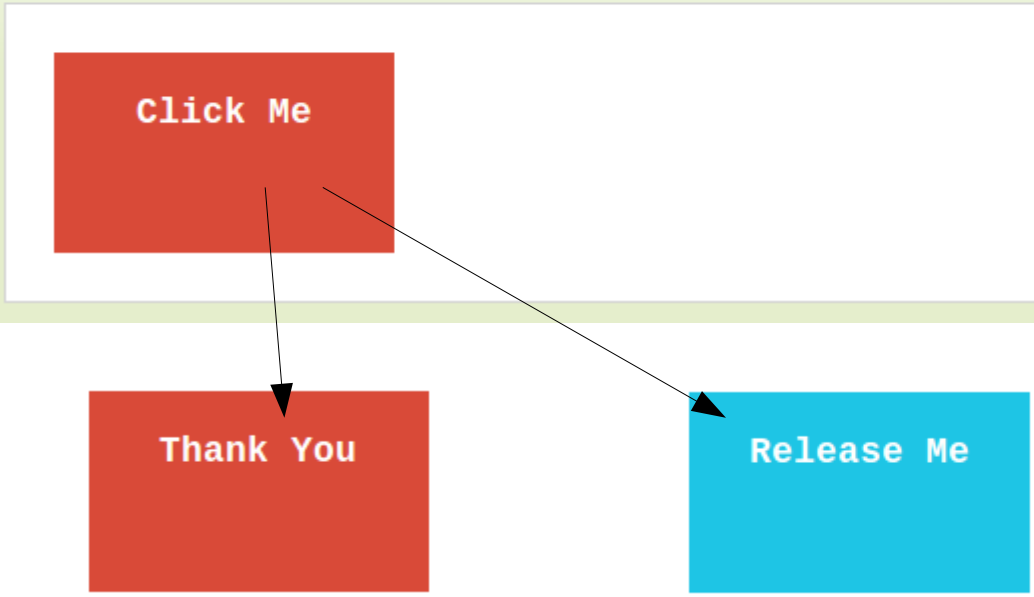
onmousedown / onmouseup

The onmousedown, onmouseup and onclick Events

The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

Example

A simple onmousedown-onmouseup example:



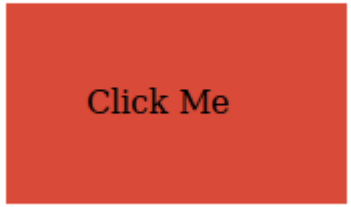
onmousedown / onmouseup

Exemplo

Source Code: Subm

```
<!DOCTYPE html>
<html>
  <body>
    <div onmousedown="mDown(this)" onmouseup="mUp(this)"
style="background-color:#D94A38;width:90px;height:20px;
padding:40px;">Click Me
    </div>

    <script>
      function mDown(obj){
        obj.style.backgroundColor="#1ec5e5";
        obj.innerHTML="Release Me"
      }

      function mUp(obj){
        obj.style.backgroundColor="#D94A38";
        obj.innerHTML="Thank You"
      }
    </script>
  </body>
</html>
```

Result:



Result:

Result:

Criando Novos Elementos HTML

Adding and Removing Nodes (HTML Elements)

Creating New HTML Elements

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

Example

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para=document.createElement("p");
var node=document.createTextNode("This is new.");
para.appendChild(node);

var element=document.getElementById("div1");
element.appendChild(para);
</script>
```

Criando Novos Elementos HTML Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <div id="div1">
      <p id="p1">This is a paragraph.</p>
      <p id="p2">This is another paragraph.</p>
    </div>

    <script>
      var para=document.createElement("p");
      var node=document.createTextNode("This is new.");
      para.appendChild(node);

      var element=document.getElementById("div1");
      element.appendChild(para);
    </script>
  </body>
</html>
```

Result:

This is a paragraph.

This is another paragraph.

This is new.

Criando Novos Elementos HTML Exemplo - Explicação

Example Explained

This code creates a new <p> element:

```
var para=document.createElement("p");
```

To add text to the <p> element, you must create a text node first. This code creates a text node:

```
var node=document.createTextNode("This is a new paragraph.");
```

Then you must append the text node to the <p> element:

```
para.appendChild(node);
```

Finally you must append the new element to an existing element.

This code finds an existing element:

```
var element=document.getElementById("div1");
```

This code appends the new element to the existing element:

```
element.appendChild(para);
```

Removendo Elementos HTML

Removing Existing HTML Elements

To remove an HTML element, you must know the parent of the element:

Example

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent=document.getElementById("div1");
var child=document.getElementById("p1");
parent.removeChild(child);
</script>
```

Removendo Elementos HTML Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <div id="div1">
      <p id="p1">This is a paragraph.</p>
      <p id="p2">This is another paragraph.</p>
    </div>

    <script>
      var parent=document.getElementById("div1");
      var child=document.getElementById("p1");
      parent.removeChild(child);
    </script>
  </body>
</html>
```

Result:

This is another paragraph.

Removendo Elementos HTML Exemplo - Explicação

Example Explained

This HTML document contains a element with two child nodes (two <p> elements):

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
```

Find the element with id="div1":

```
var parent=document.getElementById("div1");
```

Find the <p> element with id="p1":

```
var child=document.getElementById("p1");
```

Remove the child from the parent:

```
parent.removeChild(child);
```

It would be nice to be able to remove an element without referring to the parent.
But sorry. The DOM needs to know both the element you want to remove, and its parent.

Here is a common workaround: Find the child you want to remove, and use its parentNode property to find the parent:

```
var child=document.getElementById("p1");
child.parentNode.removeChild(child);
```

JS Objects

Objetos em JavaScript

"Everything" in JavaScript is an Object: a String, a Number, an Array, a Function....

In addition, JavaScript allows you to define your own objects.

JavaScript Objects

JavaScript has several **built-in** objects, like String, Date, Array, and more.

An object is just a special kind of data, with **properties** and **methods**.

Accessing Object Properties

Properties are the values associated with an object.

The syntax for accessing the property of an object is:

```
objectName.propertyName
```

This example uses the length property of the String object to find the length of a string:

```
var message="Hello World!";
var x=message.length;
```

The value of x, after execution of the code above will be:

```
12
```

Acessando os Métodos

Accessing Objects Methods

Methods are the actions that can be performed on objects.

You can call a method with the following syntax:

```
objectName.methodName()
```

This example uses the toUpperCase() method of the String object, to convert a text to uppercase:

```
var message="Hello world!";
var x=message.toUpperCase();
```

The value of x, after execution of the code above will be:

```
HELLO WORLD!
```

Criando Objetos

Creating JavaScript Objects

With JavaScript you can define and create your own objects.

There are 2 different ways to create a new object:

- 1. Define and create a direct instance of an object.
- 2. Use a function to define an object, then create new object instances.

Creating a Direct Instance

This example creates a new instance of an object, and adds four properties to it:

Example

```
person=new Object();
person.firstname="John";
person.lastname="Doe";
person.age=50;
person.eyecolor="blue";
```

Alternative syntax (using object literals):

Example

```
person={firstname:"John",lastname:"Doe",age:50,eyecolor:"blue"};
```

Criando Objetos
Exemplo 1

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var person=new Object();
      person.firstname="John";
      person.lastname="Doe";
      person.age=50;
      person.eyecolor="blue";
      document.write(person.firstname + " is " + person.age + " years
old.");
    </script>
  </body>
</html>
```

Result:

John is 50 years old.

Criando Objetos
Exemplo 2

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      person={firstname:"John",lastname:"Doe",age:50,eyecolor:"blue"}
      document.write(person.firstname + " is " + person.age + " years
old.");
    </script>
  </body>
</html>
```

Result:

John is 50 years old.

Método Construtor

Using an Object Constructor

This example uses a function to construct the object:

Example

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;
}
```

The reason for all the "this" stuff is that you're going to have more than one person at a time (which person you're dealing with must be clear). That's what "this" is: the instance of the object at hand.

Método Construtor Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      function person(firstname,lastname,age,eyecolor){
        this.firstname=firstname;
        this.lastname=lastname;
        this.age=age;
        this.eyecolor=eyecolor;
      }

      myFather=new person("John","Doe",50,"blue");

      document.write(myFather.firstname + " is " + myFather.age + "
years old.");
    </script>
  </body>
</html>
```

Result:

John is 50 years old.

Criando Instâncias do Objeto

Creating JavaScript Object Instances

Once you have a object constructor, you can create new instances of the object, like this:

```
var myFather=new person("John","Doe",50,"blue");
var myMother=new person("Sally","Rally",48,"green");
```

Criando Propriedades para o Objeto

Adding Properties to JavaScript Objects

You can add new properties to an existing object by simply giving it a value.

Assume that the personObj already exists - you can give it new properties named firstname, lastname, age, and eyecolor as follows:

```
person.firstname="John";
person.lastname="Doe";
person.age=30;
person.eyecolor="blue";

x=person.firstname;
```

The value of x, after execution of the code above will be:

```
John
```

Criando Métodos para o Objeto

Adding Methods to JavaScript Objects

Methods are just functions attached to objects.

Defining methods to an object is done inside the constructor function:

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;

this.changeName=changeName;
function changeName(name)
{
this.lastname=name;
}
}
```

The changeName() function assigns the value of name to the person's lastname property.

Criando Métodos para o Objeto Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      function person(firstname,lastname,age,eyecolor){
        this.firstname=firstname;
        this.lastname=lastname;
        this.age=age;
        this.eyecolor=eyecolor;

        this.changeName=changeName;

        function changeName(name){
          this.lastname=name;
        }
      }

      myMother=new person("Sally","Rally",48,"green");
      myMother.changeName("Doe");

      document.write(myMother.lastname);
    </script>
  </body>
</html>
```

Result:

Doe

Classes

JavaScript Classes

JavaScript is an object oriented language, but JavaScript does not use classes.

In JavaScript you don't define classes and create objects from these classes (as in most other object oriented languages).

JavaScript is prototype based, not class based.

for ... in

JavaScript for...in Loop

The JavaScript for...in statement loops through the properties of an object.

Syntax

```
for (variable in object)
  {
  code to be executed
  }
```

Note: The block of code inside of the for...in loop will be executed once for each property.

Example

Looping through the properties of an object:

Example

```
var person={fname:"John",lname:"Doe",age:25};

for (x in person)
  {
  txt=txt + person[x];
  }
```

for ... in
Exemplo

Source Code: Submit Code

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to loop through the properties of an object
named "person".</p>
    <button onclick="myFunction()">Try it</button>
    <p id="demo"></p>

    <script>
      function myFunction(){
        var x;
        var txt="";
        var person={fname:"John",lname:"Doe",age:25};

        for (x in person){
          txt=txt + person[x];
        }

        document.getElementById("demo").innerHTML=txt;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to loop through the properties of an object named "person".

Try it

Result:

Click the button to loop through the properties of an object named "person".

Try it

JohnDoe25

Objetos Números

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 pi=3.14;    // Written with decimals
var x=34;       // Written without decimals
```

Extra large or extra small numbers can be written with scientific (exponent) notation:

Example

```
var y=123e5;    // 12300000
var z=123e-5;   // 0.00123
```

All JavaScript Numbers are 64-bit

JavaScript is not a typed language. Unlike many other programming languages, it does not define different types of numbers, like integers, short, long, floating-point etc.

All numbers in JavaScript are stored as 64-bit (8-bytes) base 10, floating point numbers.

Precisão Numérica

Precision

Integers (numbers without a period or exponent notation) are considered accurate up to 15 digits.

The maximum number of decimals is 17, but floating point arithmetic is not always 100% accurate:

Example

```
var x=0.2+0.1;
```

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var x;
      document.write("<p>Only 17 digits: ");
      x=12345678901234567890;
      document.write(x + "</p>");

      document.write("<p>0.2 + 0.1 = ");
      x=0.2+0.1;
      document.write(x + "</p>");

      document.write("<p>It helps multiplying and dividing by 10: ");
      x=(0.2*10+0.1*10)/10;
      document.write(x +"</p>");
    </script>
  </body>
</html>
```

Result:

Only 17 digits: 12345678901234567000

0.2 + 0.1 = 0.30000000000000004

It helps multiplying and dividing by 10: 0.3

Base Octal e Hexadecimal

Octal and Hexadecimal

JavaScript interprets numeric constants as octal if they are preceded by a zero, and as hexadecimal if they are preceded by a zero and "x."

Example

```
var y=0377;
var z=0xFF;
```

Never write a number with a leading zero, unless you want an octal conversion.

Números: Propriedades e Métodos

Number Properties and Methods

Properties:

- MAX VALUE
- MIN VALUE
- NEGATIVE INFINITIVE
- POSITIVE INFINITIVE
- NaN
- prototype
- constructor

Methods:

- toExponential()
- toFixed()
- toPrecision()
- toString()
- valueOf()

Objeto *String*

The String object is used for storing and manipulating text.

JavaScript Strings

A string simply stores a series of characters like "John Doe".

A string can be any text inside quotes. You can use simple or double quotes:

Example

```
var carname="Volvo XC60";
var carname='Volvo XC60';
```

You can access each character in a string with its position (index):

Example

```
var character=carname[7];
```

String indexes are zero-based, which means the first character is [0], the second is [1], and so on.

String Exemplo

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\"";
```

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <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.write(carname1 + "<br>")
      document.write(carname2 + "<br>")
      document.write(answer1 + "<br>")
      document.write(answer2 + "<br>")
      document.write(answer3 + "<br>")
    </script>
  </body>
</html>
```

Result:

Volvo XC60
Volvo XC60
It's alright
He is called 'Johnny'
He is called "Johnny"

Tamanho da *String*

String Length

The length of a string (a string object) is found in the built in property **length**:

Example

```
var txt="Hello World!";
document.write(txt.length);

var txt="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
document.write(txt.length);
```

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var txt = "Hello World!";
      document.write("<p>" + txt.length + "</p>");

      var txt="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      document.write("<p>" + txt.length + "</p>");
    </script>
  </body>
</html>
```

Result:

12

26

Descobrindo uma *String* numa *String*

Finding a String in a String

The **indexOf()** method returns the position (as a number) of the first found occurrence of a specified text inside a string:

Example

```
var str="Hello world, welcome to the universe.";
var n=str.indexOf("welcome");
```

The method returns -1 if the specified text is not found.

The lastIndexOf() method starts searching at the end of the string instead of at the beginning.

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p id="p1">Click the button to locate where "locate" first occurs.
</p>
    <p id="p2">0</p>
    <button onclick="myFunction()">Try it</button>

    <script>
      function myFunction(){
        var str=document.getElementById("p1").innerHTML;
        var n=str.indexOf("locate");
        document.getElementById("p2").innerHTML=n+1;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to locate where "locate" first occurs.

0

Try it

Comparando Conteúdo

Matching Content

The **match()** method can be used to search for a matching content in a string:

Example

```
var str="Hello world!";
document.write(str.match("world") + "<br>");
document.write(str.match("World") + "<br>");
document.write(str.match("world!"));
```

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var str="Hello world!";
      document.write(str.match("world") + "<br>");
      document.write(str.match("World") + "<br>");
      document.write(str.match("world!"));
    </script>
  </body>
</html>
```

Result:

world
null
world!

Sobrescrevendo Conteúdo

Replacing 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");
```

Source Code: Submit Code

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to replace "Microsoft" with "W3Schools" in
the paragraph below:</p>
    <p id="demo">Please visit Microsoft!</p>

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

    <script>
      function myFunction(){
        var str=document.getElementById("demo").innerHTML;
        var n=str.replace("Microsoft","W3Schools");
        document.getElementById("demo").innerHTML=n;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to replace "Microsoft" with "W3Schools" in the paragraph below:

Please visit Microsoft!

Try it

Result:

Click the button to replace "Microsoft" with "W3Schools" in the paragraph below:

Please visit W3Schools!

Try it

Caixa Alta e Caixa Baixa

Upper Case and Lower Case

A string is converted to upper/lower case with the methods **toUpperCase()** / **toLowerCase()**:

Example

```
var txt="Hello World!";       // String
var txt1=txt.toUpperCase();   // txt1 is txt converted to upper
var txt2=txt.toLowerCase();   // txt2 is txt converted to lower
```

Source Code: Subr

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var txt="Hello World!";
      document.write("<p>" + txt.toUpperCase() + "</p>");
      document.write("<p>" + txt.toLowerCase() + "</p>");
      document.write("<p>" + txt + "</p>");
    </script>
    <p>
      The methods returns a new string.
      The original string is not changed.
    </p>
  </body>
</html>
```

Result:

HELLO WORLD!

hello world!

Hello World!

The methods returns a new string. The original string is not changed.

Convertendo uma *String*

Convert a String to an Array

A string is converted to an array with the built in method string.**split()**:

Example

```
txt="a,b,c,d,e"   // String
txt.split(",");   // Split on commas
txt.split(" ");   // Split on spaces
txt.split("|");   // Split on pipe
```

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p id="demo">Click the button to display the array values after
the split.</p>
    <button onclick="myFunction()">Try it</button>

    <script>
      function myFunction(){
        var str="a,b,c,d,e,f";
        var n=str.split(",");
        document.getElementById("demo").innerHTML=n[0];
      }
    </script>
  </body>
</html>
```

Result:

Click the button to display the array values after the split.

Try it

Result:

a

Try it

Caracteres Especiais

Special Characters

The backslash (\) can be used to insert apostrophes, new lines, quotes, and other special characters into a string.

Look at the following JavaScript code:

```
var txt="We are the so-called "Vikings" from the north.";
document.write(txt);
```

In JavaScript, a string is started and stopped with either single or double quotes. This means that the string above will be chopped to: We are the so-called

To solve this problem, you must place a backslash (\) before each double quote in "Viking". This turns each double quote into a string literal:

```
var txt="We are the so-called \"Vikings\" from the north.";
document.write(txt);
```

JavaScript will now output the proper text string: We are the so-called "Vikings" from the north.

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

String: Propriedades e Métodos

String Properties and Methods

Properties:

- length
- prototype
- constructor

Methods:

- charAt()
- charCodeAt()
- concat()
- fromCharCode()
- indexOf()
- lastIndexOf()
- match()
- replace()
- search()
- slice()
- split()
- substr()
- substring()
- toLowerCase()
- toUpperCase()
- valueOf()

Objeto *Date*

- Objeto utilizado para se trabalhar com data e hora.

Create a Date Object

The Date object is used to work with dates and times.

Date objects are created with the Date() constructor.

There are four ways of initiating a date:

```
new Date() // current date and time
new Date(milliseconds) //milliseconds since 1970/01/01
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

Most parameters above are optional. Not specifying, causes 0 to be passed in.

Once a Date object is created, a number of methods allow you to operate on it. Most methods allow you to get and set the year, month, day, hour, minute, second, and milliseconds of the object, using either local time or UTC (universal, or GMT) time.

All dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC) with a day containing 86,400,000 milliseconds.

Some examples of initiating a date:

```
var today = new Date()
var d1 = new Date("October 13, 1975 11:13:00")
var d2 = new Date(79,5,24)
var d3 = new Date(79,5,24,11,33,0)
```

Configurando uma Data

Set Dates

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

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

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

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

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

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

Comparando Duas Datas

Compare Two Dates

The Date object is also used to compare two dates.

The following example compares today's date with the 14th January 2100:

```
var x=new Date();
x.setFullYear(2100,0,14);
var today = new Date();

if (x>today)
  {
  alert("Today is before 14th January 2100");
  }
else
  {
  alert("Today is after 14th January 2100");
  }
```

Objeto *Array*

The Array object is used to store multiple values in a single variable.

Create an array, and assign values to it:

Example

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

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.

Objeto *Array*
Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var i;
      var mycars = new Array();

      mycars[0] = "Saab";
      mycars[1] = "Volvo";
      mycars[2] = "BMW";

      for (i=0;i<mycars.length;i++){
        document.write(mycars[i] + "<br>");
      }
    </script>
  </body>
</html>
```

Result:

Saab
Volvo
BMW

Criando um Vetor (*Array*)

Create an Array

An array can be created in three ways.

The following code creates an Array object called myCars:

1: Regular:

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

2: Condensed:

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

3: Literal:

```
var myCars=["Saab","Volvo","BMW"];
```

Acessando a Posição de um Vetor (*Array*)

Access an Array

You refer to an element in an array by referring to the **index** number.

This statement access the value of the first element in myCars:

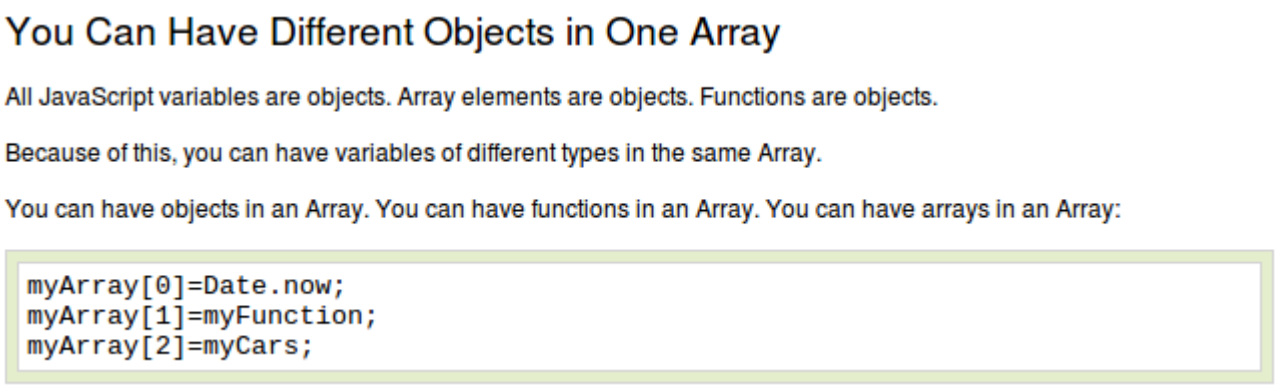
```
var name=myCars[0];
```

This statement modifies the first element in myCars:

```
myCars[0]="Opel";
```

[0] is the first element in an array. [1] is the second . . . . . (indexes start with 0)

Diferentes Tipos de Objetos no Mesmo Vetor (*Array*)



You Can Have Different Objects in One Array

All JavaScript variables are objects. Array elements are objects. Functions are 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;
```

Vetor (*Array*): Métodos e Propriedades

Array Methods and Properties

The Array object has predefined properties and methods:

```
var x=myCars.length              // the number of elements in myCars
var y=myCars.indexOf("Volvo")    // the index position of "Volvo"
```

Complete Array Object Reference

For a complete reference of all properties and methods, go to our complete Array object reference.

The reference contains a description (and more examples) of all Array properties and methods.

Complete array object reference.

Vetor (*Array*): Criando Novos Métodos

Create New Methods

Prototype is a global constructor in JavaScript. It can construct new properties and methods for any JavaScript Objects.

Example: Make a new Array method.

```
Array.prototype.ucase=function()
{
  for (i=0;i<this.length;i++)
  {this[i]=this[i].toUpperCase();}
}
```

The example above makes a new array method that transforms array values into upper case.

Vetor (*Array*): Criando Novos Métodos

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p id="demo">Click the button to create an array, call the new
ucase() method, and display the result.</p>
    <button onclick="myFunction()">Try it</button>

    <script>
      Array.prototype.myUcase=function(){
        for (i=0;i<this.length;i++){
          this[i]=this[i].toUpperCase();
        }
      }

      function myFunction(){
        var fruits = ["Banana", "Orange", "Apple", "Mango"];
        fruits.myUcase();
        var x=document.getElementById("demo");
        x.innerHTML=fruits;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to create an array, call the new ucase() method, and display the result.

Try it

Result:

BANANA,ORANGE,APPLE,MANGO

Try it

Objeto Booleano

Create a Boolean Object

The Boolean object represents two values: "true" or "false".

The following code creates a Boolean object called myBoolean:

```
var myBoolean=new Boolean();
```

If the Boolean object has no initial value, or if the passed value is one of the following:

- 0
- -0
- null
- ""
- false
- undefined
- NaN

the object is set to false. For any other value it is set to true (even with the string "false")!

Objeto Matemático

Math Object

The Math object allows you to perform mathematical tasks.

The Math object includes several mathematical constants and methods.

Syntax for using properties/methods of Math:

```
var x=Math.PI;
var y=Math.sqrt(16);
```

Note: Math is not a constructor. All properties and methods of Math can be called by using Math as an object without creating it.

Mathematical Constants

JavaScript provides eight mathematical constants that can be accessed from the Math object. These are: E, PI, square root of 2, square root of 1/2, natural log of 2, natural log of 10, base-2 log of E, and base-10 log of E.

You may reference these constants from your JavaScript like this:

```
Math.E
Math.PI
Math.SQRT2
Math.SQRT1_2
Math.LN2
Math.LN10
Math.LOG2E
Math.LOG10E
```

Métodos do Objeto Matemático

Mathematical Methods

In addition to the mathematical constants that can be accessed from the Math object there are also several methods available.

The following example uses the round() method of the Math object to round a number to the nearest integer:

```
document.write(Math.round(4.7));
```

The code above will result in the following output:

```
5
```

The following example uses the random() method of the Math object to return a random number between 0 and 1:

```
document.write(Math.random());
```

The code above can result in the following output:

```
0.6486976376135797
```

The following example uses the floor() and random() methods of the Math object to return a random number between 0 and 10:

```
document.write(Math.floor(Math.random()*11));
```

The code above can result in the following output:

```
0
```

Objeto Expressões Regulares

What is RegExp?

A regular expression is an object that describes a pattern of characters.

When you search in a text, you can use a pattern to describe what you are searching for.

A simple pattern can be one single character.

A more complicated pattern can consist of more characters, and can be used for parsing, format checking, substitution and more.

Regular expressions are used to perform powerful pattern-matching and "search-and-replace" functions on text.

Syntax

```
var patt=new RegExp(pattern,modifiers);

or more simply:

var patt=/pattern/modifiers;
```

- pattern specifies the pattern of an expression
- modifiers specify if a search should be global, case-sensitive, etc.

Expressões Regulares
Modificadores

RegExp Modifiers

Modifiers are used to perform case-insensitive and global searches.

The i modifier is used to perform case-insensitive matching.

The g modifier is used to perform a global match (find all matches rather than stopping after the first match).

Example 1

Do a case-insensitive search for "w3schools" in a string:

```
var str="Visit W3Schools";
var patt1=/w3schools/i;
```

The marked text below shows where the expression gets a match:

```
Visit W3Schools
```

Expressões Regulares
Exemplo 1

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var str = "Visit W3Schools";
      var patt1 = /w3schools/i;
      document.write(str.match(patt1));
    </script>
  </body>
</html>
```

Result:

W3Schools

Expressões Regulares Exemplo 2

Example 2

Do a global search for "is":

```
var str="Is this all there is?";
var patt1=/is/g;
```

The marked text below shows where the expression gets a match:

```
Is this all there is?
```

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var str="Is this all there is?";
       var patt1=/is/g;
       document.write(str.match(patt1));
    </script>
  </body>
</html>
```

Result:

is,is

Expressões Regulares Exemplo 3

Example 3

Do a global, case-insensitive search for "is":

```
var str="Is this all there is?";
var patt1=/is/gi;
```

The marked text below shows where the expression gets a match:

```
Is this all there is?
```

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var str="Is this all there is?";
      var patt1=/is/gi;
      document.write(str.match(patt1));
    </script>
   </body>
</html>
```

Result:

Is,is,is

Expressões Regulares
Método *test()*

test()

The test() method searches a string for a specified value, and returns true or false, depending on the result.

The following example searches a string for the character "e":

Example

```
var patt1=new RegExp("e");
document.write(patt1.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
```

Expressões Regulares
Método *test()* - Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var patt1=new RegExp("e");

      document.write(patt1.test("The best things in life are free"));
    </script>
  </body>
</html>
```

Result:

true

Expressões Regulares
Método *exec()*

exec()

The exec() method searches a string for a specified value, and returns the text of the found value. If no match is found, it returns *null*.

The following example searches a string for the character "e":

Example 1

```
var patt1=new RegExp("e");
document.write(patt1.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
```

Expressões Regulares
Método *exec()* - Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      var patt1=new RegExp("e");

      document.write(patt1.exec("The best things in life are free"));
    </script>
  </body>
</html>
```

Result:

e

JS Window

Window (Janela)

The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.

The Browser Object Model (BOM)

There are no official standards for the **B**rowser **O**bject **M**odel (BOM).

Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM.

The Window Object

The **window** object is supported by all browsers. It represent the browsers window.

All global JavaScript objects, functions, and variables automatically become members of the window object.

Global variables are properties of the window object.

Global functions are methods of the window object.

Even the document object (of the HTML DOM) is a property of the window object:

```
window.document.getElementById("header");
```

is the same as:

```
document.getElementById("header");
```

Tamanho da Janela (*Window*)

Window Size

Three different properties can be used to determine the size of the browser window (the browser viewport, NOT including toolbars and scrollbars).

For Internet Explorer, Chrome, Firefox, Opera, and Safari:

- window.innerHeight - the inner height of the browser window
- window.innerWidth - the inner width of the browser window

For Internet Explorer 8, 7, 6, 5:

- document.documentElement.clientHeight
- document.documentElement.clientWidth
- or
- document.body.clientHeight
- document.body.clientWidth

A practical JavaScript solution (covering all browsers):

Example

```
var w=window.innerWidth
|| document.documentElement.clientWidth
|| document.body.clientWidth;

var h=window.innerHeight
|| document.documentElement.clientHeight
|| document.body.clientHeight;
```

The example displays the browser window's height and width: (NOT including toolbars/scrollbars)

Tamanho da Janela (*Window*) Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p id="demo"></p>

    <script>
      var w=window.innerWidth
      || document.documentElement.clientWidth
      || document.body.clientWidth;

      var h=window.innerHeight
      || document.documentElement.clientHeight
      || document.body.clientHeight;

      x=document.getElementById("demo");
      x.innerHTML="Browser inner window width: " + w + ", height: " +
h + "."
    </script>
  </body>
</html>
```

Result:

Browser inner window width: 494, height: 473.

Outros Métodos de Janela

Other Window Methods

Some other methods:

- window.open() - open a new window
- window.close() - close the current window
- window.moveTo() -move the current window
- window.resizeTo() -resize the current window

Tela da Janela (*Window Screen*)

Window Screen

The **window.screen** object can be written without the window prefix.

Some properties:

- screen.availWidth - available screen width
- screen.availHeight - available screen height

Window Screen Available Width

The screen.availWidth property returns the width of the visitor's screen, in pixels, minus interface features like the Windows Taskbar.

Example

Return the available width of your screen:

```
<script>

document.write("Available Width: " + screen.availWidth);

</script>
```

The output of the code above will be:

```
Available Width: 1024
```

Tela da Janela (*Window Screen*) Exemplo

Source Code: Submit C

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      document.write("Available Width: " + screen.availWidth);
    </script>
  </body>
</html>
```

Result:

Available Width: 1024

Todas as Propriedades de Tela em um Único Exemplo

Source Code: Submit Code

```
<!DOCTYPE html>
<html>
  <body>
    <h2>Todas as Propriedades de Tela em um Único Exemplo<h2>
    <h3>Your Screen:</h3>

    <script>
      document.write("Total width/height: ");
      document.write(screen.width + "*" + screen.height);
      document.write("<br>");
      document.write("Available width/height: ");
      document.write(screen.availWidth + "*" + screen.availHeight);
      document.write("<br>");
      document.write("Color depth: ");
      document.write(screen.colorDepth);
      document.write("<br>");
      document.write("Color resolution: ");
      document.write(screen.pixelDepth);
    </script>
  </body>
</html>
```

Result:

Todas as Propriedades de Tela em um Único Exemplo

Your Screen:

Total width/height: 1024*768
Available width/height: 1024*744
Color depth: 24
Color resolution: 24

Localização da Janela (*Window Location*)

The window.location object can be used to get the current page address (URL) and to redirect the browser to a new page.

Window Location

The **window.location** object can be written without the window prefix.

Some examples:

- location.hostname returns the domain name of the web host
- location.path returns the path and filename of the current page
- location.port returns the port of the web host (80 or 443)
- location protocol returns the web protocol used (http:// or https://)

Window Location - Href

Window Location Href

The location.href property returns the URL of the current page.

Example

Return the entire URL (of the current page):

```
<script>

document.write(location.href);

</script>
```

The output of the code above is:

```
http://www.w3schools.com/js/js_window_location.asp
```

Window Location - Pathname

Window Location Pathname

The location.pathname property returns the path name of a URL.

Example

Return the path name of the current URL:

```
<script>

document.write(location.pathname);

</script>
```

The output of the code above is:

```
/js/js_window_location.asp
```

Window Location - Assign

Window Location Assign

The location.assign() method loads a new document.

Example

Load a new document:

```
<html>
<head>
<script>
function newDoc()
  {
  window.location.assign("http://www.w3schools.com")
  }
</script>
</head>
<body>

<input type="button" value="Load new document" onclick="newDoc()">

</body>
</html>
```

Window Location – Assign
Exemplo

Result:

Load new document

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function newDoc() {
        window.location.assign("http://www.w3schools.com")
      }
    </script>
  </head>
  <body>
    <input type="button" value="Load new document" onclick="newDoc()">
  </body>
</html>
```

Result:

Window History

The window.history object contains the browsers history.

Window History

The **window.history** object can be written without the window prefix.

To protect the privacy of the users, there are limitations to how JavaScript can access this object.

Some methods:

- history.back() - same as clicking back in the browser
- history.forward() - same as clicking forward in the browser

Window History Back

Window History Back

The history.back() method loads the previous URL in the history list.

This is the same as clicking the Back button in the browser.

Example

Create a back button on a page:

```
<html>
<head>
<script>
function goBack()
  {
  window.history.back()
  }
</script>
</head>
<body>

<input type="button" value="Back" onclick="goBack()">

</body>
</html>
```

The output of the code above will be:

Back

Window History Forward

Window History Forward

The history forward() method loads the next URL in the history list.

This is the same as clicking the Forward button in the browser.

Example

Create a forward button on a page:

```
<html>
<head>
<script>
function goForward()
  {
  window.history.forward()
  }
</script>
</head>
<body>

<input type="button" value="Forward" onclick="goForward()">

</body>
</html>
```

The output of the code above will be:

Forward

Window Navigator

The window.navigator object contains information about the visitor's browser.

Window Navigator

The **window.navigator** object can be written without the window prefix.

Example

```
<div id="example"></div>

<script>

txt = "<p>Browser CodeName: " + navigator.appCodeName + "</p>";
txt+= "<p>Browser Name: " + navigator.appName + "</p>";
txt+= "<p>Browser Version: " + navigator.appVersion + "</p>";
txt+= "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>";
txt+= "<p>Platform: " + navigator.platform + "</p>";
txt+= "<p>User-agent header: " + navigator.userAgent + "</p>";
txt+= "<p>User-agent language: " + navigator.systemLanguage + "</p>";

document.getElementById("example").innerHTML=txt;

</script>
```

Window Navigator
Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <div id="example"></div>

    <script>
      txt = "<p>Browser CodeName: " + navigator.appCodeName + "</p>";
      txt+= "<p>Browser Name: " + navigator.appName + "</p>";
      txt+= "<p>Browser Version: " + navigator.appVersion + "</p>";
      txt+= "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>";
      txt+= "<p>Platform: " + navigator.platform + "</p>";
      txt+= "<p>User-agent header: " + navigator.userAgent + "</p>";
      txt+= "<p>User-agent language: " + navigator.systemLanguage +
"</p>";

      document.getElementById("example").innerHTML=txt;
    </script>
  </body>
</html>
```

Result:

Browser CodeName: Mozilla

Browser Name: Netscape

Browser Version: 5.0 (X11)

Cookies Enabled: true

Platform: Linux i686

User-agent header: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:18.0) Gecko/20100101 Firefox/18.0

User-agent language: undefined

Atenção!

Warning !!!

The information from the navigator object can often be misleading, and should not be used to detect browser versions because:

- The navigator data can be changed by the browser owner
- Some browsers misidentify themselves to bypass site tests
- Browsers cannot report new operating systems, released later than the browser

---

Browser Detection

Since the navigator object can be misleading about browser detection, using object detection can be used to sniff out different browsers.

Since different browsers support different objects, you can use objects to detect browsers. For example, since only Opera supports the property "window.opera", you can use that to identify Opera.

Example: if (window.opera) {...some action...}

Alert Box

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
window.alert("sometext");
```

The **window.alert** method can be written without the window prefix.

Example

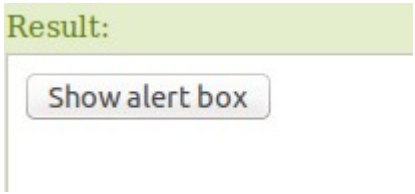
```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction()
{
alert("I am an alert box!");
}
</script>
</head>
<body>

<input type="button" onclick="myFunction()" value="Show alert box">

</body>
</html>
```

Alert Box
Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function myFunction(){
        alert("Hello! I am an alert box!");
      }
    </script>
  </head>
  <body>
    <input type="button" onclick="myFunction()" value="Show alert
box" />
  </body>
</html>
```

Result:



Confirm Box

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

```
window.confirm("sometext");
```

The **window.confirm()** method can be written without the window prefix.

Example

```
var r=confirm("Press a button");
if (r==true)
  {
  x="You pressed OK!";
  }
else
  {
  x="You pressed Cancel!";
  }
```

Confirm Box
Exemplo

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to display a confirm box.</p>
    <button onclick="myFunction()">Try it</button>

    <p id="demo"></p>

    <script>
      function myFunction(){
        var x;
        var r=confirm("Press a button!");
        if (r==true){
          x="You pressed OK!";
        }
        else{
          x="You pressed Cancel!";
        }
        document.getElementById("demo").innerHTML=x;
      }
    </script>
  </body>
</html>
```

Result:

Click the button to display a confirm box.

Try it

Result:

Click the button to display a confirm box.

Try it

You pressed OK!

Prompt Box

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
window.prompt("sometext","defaultvalue");
```

The **window.prompt()** method can be written without the window prefix.

Example

```
var name=prompt("Please enter your name","Harry Potter");
if (name!=null && name!="")
  {
  x="Hello " + name + "! How are you today?";
  }
```

Prompt Box
Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to demonstrate the prompt box.</p>
    <button onclick="myFunction()">Try it</button>

    <p id="demo"></p>

    <script>
      function myFunction(){
        var x;
        var name=prompt("Please enter your name","Harry Potter");

        if (name!=null){
          x="Hello " + name + "! How are you today?";
          document.getElementById("demo").innerHTML=x;
        }
      }
    </script>
  </body>
</html>
```

Result:

Click the button to demonstrate the prompt box.

Try it

Result:

Click the button to demonstrate the prompt box.

Try it

Hello Edwar Saliba Júnior! How are you today?

Quebra de Linha

Line Breaks

To display line breaks inside a popup box, use a back-slash followed by the character n.

Example

```
alert("Hello\nHow are you?");
```

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to demonstrate line-breaks in a popup box.</p>
    <button onclick="myFunction()">Try it</button>

    <p id="demo"></p>

    <script>
      function myFunction(){
        alert("Hello\nHow are you?");
      }
    </script>
  </body>
</html>
```

Result:

Click the button to demonstrate line-breaks in a popup box.

Try it

Timing Events

JavaScript can be executed in time-intervals.

This is called timing events.

JavaScript Timing Events

With JavaScript, it is possible to execute some code at specified time-intervals. This is called timing events.

It's very easy to time events in JavaScript. The two key methods that are used are:

- setInterval() - executes a function, over and over again, at specified time intervals
- setTimeout() - executes a function, once, after waiting a specified number of milliseconds

Note: The setInterval() and setTimeout() are both methods of the HTML DOM Window object.

Método *setInterval()*

The setInterval() Method

The setInterval() method will wait a specified number of milliseconds, and then execute a specified function, and it will continue to execute the function, once at every given time-interval.

Syntax

```
window.setInterval("javascript function",milliseconds);
```

The **window.setInterval()** method can be written without the window prefix.

The first parameter of setInterval() should be a function.

The second parameter indicates the length of the time-intervals between each execution.

Note: There are 1000 milliseconds in one second.

Example

Alert "hello" every 3 seconds:

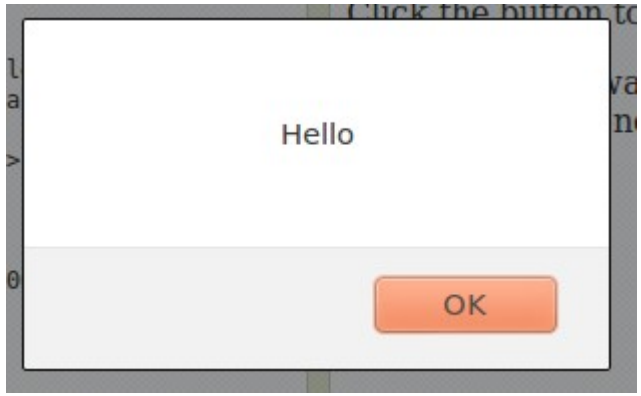
```
setInterval(function(){alert("Hello")},3000);
```

The example show you how the setInterval() method works, but it is not very likely that you want to alert a message every 3 seconds.

Método *setInterval()* - Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to wait 3 seconds, then alert "Hello".</p>
    <p>After clicking away the alert box, an new alert box will
appear in 3 seconds. This goes on forever...</p>
    <button onclick="myFunction()">Try it</button>

    <script>
      function myFunction(){
        setInterval(function(){alert("Hello")},3000);
      }
    </script>
  </body>
</html>
```

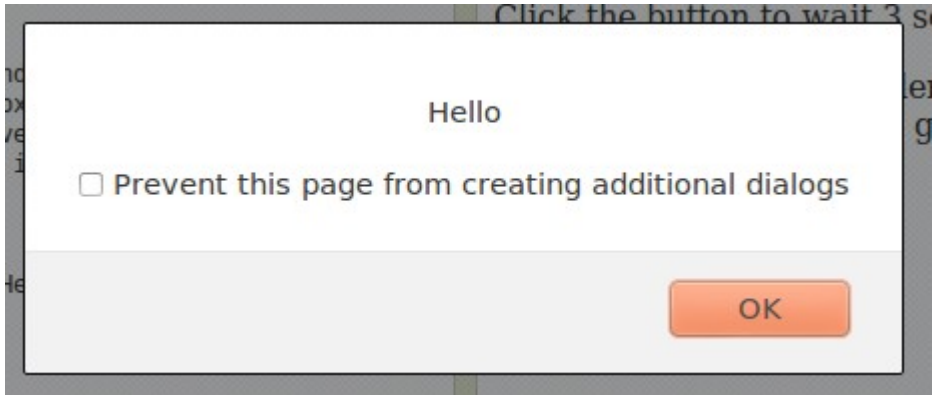


Result:

Click the button to wait 3 seconds, then alert "Hello".

After clicking away the alert box, an new alert box will appear in 3 seconds. This goes on forever...

Try it



Método *setInterval()* - Exemplo 2

Below is an example that will display the current time. The setInterval() method is used to execute the function once every 1 second, just like a digital watch.

Example

Display the current time:

```
var myVar=setInterval(function(){myTimer()},1000);

function myTimer()
{
var d=new Date();
var t=d.toLocaleTimeString();
document.getElementById("demo").innerHTML=t;
}
```

Source Code:

```
<!DOCTYPE html>
<html>
  <body>
    <p>A script on this page starts this clock:</p>
    <p id="demo"></p>

    <script>
      var myVar=setInterval(function(){myTimer()},1000);

      function myTimer(){
        var d=new Date();
        var t=d.toLocaleTimeString();
        document.getElementById("demo").innerHTML=t;
      }
    </script>
  </body>
</html>
```

Result:

A script on this page starts this clock:

12:00:46 AM

Como Parar a Execu-ção

How to Stop the Execution?

The clearInterval() method is used to stop further executions of the function specified in the setInterval() method.

Syntax

```
window.clearInterval(intervalVariable)
```

The **window.clearInterval()** method can be written without the window prefix.

To be able to use the clearInterval() method, you must use a global variable when creating the interval method:

```
myVar=setInterval("javascript function",milliseconds);
```

Then you will be able to stop the execution by calling the clearInterval() method.

Example

Same example as above, but we have added a "Stop time" button:

```
<p id="demo"></p>
<button onclick="myStopFunction()">Stop time</button>

<script>
var myVar=setInterval(function(){myTimer()},1000);
function myTimer()
{
var d=new Date();
var t=d.toLocaleTimeString();
document.getElementById("demo").innerHTML=t;
}
function myStopFunction()
{
clearInterval(myVar);
}
</script>
```

Como Parar a Execução Exemplo

Source Code: Subn

```
<!DOCTYPE html>
<html>
  <body>
    <p>A script on this page starts this clock:</p>
    <p id="demo"></p>
    <button onclick="myStopFunction()">Stop time</button>

    <script>
      var myVar=setInterval(function(){myTimer()},1000);
      function myTimer(){
        var d=new Date();
        var t=d.toLocaleTimeString();
        document.getElementById("demo").innerHTML=t;
      }

      function myStopFunction(){
        clearInterval(myVar);
      }
    </script>
  </body>
</html>
```

Result:

A script on this page starts this clock:

12:05:35 AM

Stop time

Result:

A script on this page starts this clock:

12:06:30 AM

Stop time

Método: *setTimeout()*

The setTimeout() Method

Syntax

```
window.setTimeout("javascript function",milliseconds);
```

The **window.setTimeout()** method can be written without the window prefix.

The setTimeout() method will wait the specified number of milliseconds, and then execute the specified function.

The first parameter of setTimeout() should be a function.

The second parameter indicates how many milliseconds, from now, you want to execute the first parameter.

Example

Wait 3 seconds, then alert "Hello":

```
setTimeout(function(){alert("Hello")},3000);
```

Método: *setTimeout()* Exemplo

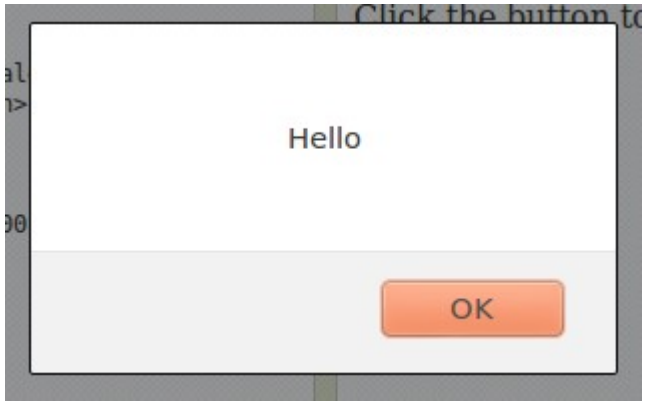
Source Code: Submit Code ›

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the button to wait 3 seconds, then alert "Hello".</p>
    <button onclick="myFunction()">Try it</button>

    <script>
      function myFunction(){
        setTimeout(function(){alert("Hello")},3000);
      }
    </script>
  </body>
</html>
```

Result:

Click the button to wait 3 seconds, then alert "Hello".

Try it



Como Para a Execução?

How to Stop the Execution?

The clearTimeout() method is used to stop the execution of the function specified in the setTimeout() method.

Syntax

```
window.clearTimeout(timeoutVariable)
```

The **window.clearTimeout()** method can be written without the window prefix.

To be able to use the clearTimeout() method, you must use a global variable when creating the timeout method:

```
myVar=setTimeout("javascript function",milliseconds);
```

Then, if the function has not already been executed, you will be able to stop the execution by calling the clearTimeout() method.

Example

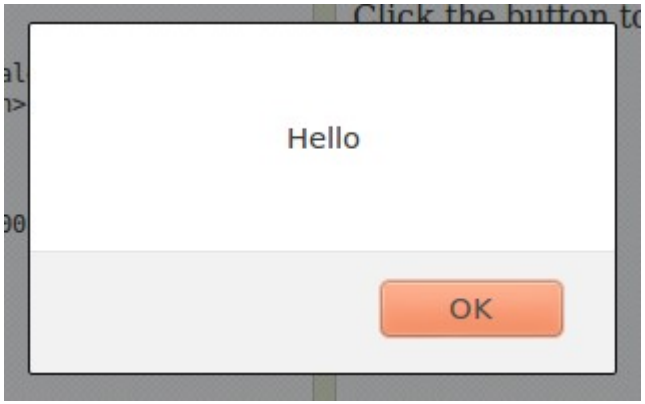
Same example as above, but we have added a "Stop the alert" button:

```
var myVar;

function myFunction()
{
myVar=setTimeout(function(){alert("Hello")},3000);
}

function myStopFunction()
{
clearTimeout(myVar);
}
```

Como Parar a Execução - Exemplo

Source Code: Submit Code »

```
<!DOCTYPE html>
<html>
  <body>
    <p>Click the first button alert "Hello" after waiting 3 seconds.
</p>
    <p>Click the second button to prevent the first function to
execute. (You must click it before the 3 seconds are up.)</p>
    <button onclick="myFunction()">Try it</button>
    <button onclick="myStopFunction()">Stop the alert</button>

    <script>
      var myVar;

      function myFunction(){
        myVar=setTimeout(function(){alert("Hello")},3000);
      }

      function myStopFunction(){
        clearTimeout(myVar);
      }
    </script>
  </body>
</html>
```



Result:

Click the first button alert "Hello" after waiting 3 seconds.

Click the second button to prevent the first function to execute. (You must click it before the 3 seconds are up.)

Try it | Stop the alert

Cookies - Introdução

A cookie is often used to identify a user.

What is a Cookie?

A cookie is a variable that is stored on the visitor's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With JavaScript, you can both create and retrieve cookie values.

Examples of cookies:

- Name cookie - The first time a visitor arrives to your web page, he or she must fill in her/his name. The name is then stored in a cookie. Next time the visitor arrives at your page, he or she could get a welcome message like "Welcome John Doe!" The name is retrieved from the stored cookie
- Date cookie - The first time a visitor arrives to your web page, the current date is stored in a cookie. Next time the visitor arrives at your page, he or she could get a message like "Your last visit was on Tuesday August 11, 2005!" The date is retrieved from the stored cookie

Criando e Armazenando um *Cookie* Simples – Parte 01

Create and Store a Cookie

In this example we will create a cookie that stores the name of a visitor. The first time a visitor arrives to the web page, he or she will be asked to fill in her/his name. The name is then stored in a cookie. The next time the visitor arrives at the same page, he or she will get welcome message.

First, we create a function that stores the name of the visitor in a cookie variable:

```
function setCookie(c_name,value,exdays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate() + exdays);
var c_value=escape(value) + ((exdays==null) ? "" : ";
expires="+exdate.toUTCString());
document.cookie=c_name + "=" + c_value;
}
```

The parameters of the function above hold the name of the cookie, the value of the cookie, and the number of days until the cookie expires.

In the function above we first convert the number of days to a valid date, then we add the number of days until the cookie should expire. After that we store the cookie name, cookie value and the expiration date in the document.cookie object.

Criando e Armazenando um *Cookie* Simples – Parte 02

Then, we create another function that returns a specified cookie:

```
function getCookie(c_name)
{
var i,x,y,ARRcookies=document.cookie.split(";");
for (i=0;i<ARRcookies.length;i++)
{
  x=ARRcookies[i].substr(0,ARRcookies[i].indexOf("="));
  y=ARRcookies[i].substr(ARRcookies[i].indexOf("=")+1);
  x=x.replace(/^\s+|\s+$/g,"");
  if (x==c_name)
    {
    return unescape(y);
    }
  }
}
```

The function above makes an array to retrieve cookie names and values, then it checks if the specified cookie exists, and returns the cookie value.

Criando e Armazenando um *Cookie* Simples – Parte 03

Last, we create the function that displays a welcome message if the cookie is set, and if the cookie is not set it will display a prompt box, asking for the name of the user, and stores the username cookie for 365 days, by calling the setCookie function:

```
function checkCookie()
{
var username=getCookie("username");
  if (username!=null && username!="")
  {
  alert("Welcome again " + username);
  }
else
  {
  username=prompt("Please enter your name:","");
  if (username!=null && username!="")
    {
    setCookie("username",username,365);
    }
  }
}
```

Bibliografia

- W3Schools. **JavaScript**. Disponível em: <http://www.w3schools.com/js/default.asp> Acesso em: 06 jan. 2013.