Programação e Serviços Web
JavaScript

Programação e Serviços Web
1º Ano 2º Semestre

Cofinanciado por:

Programação e Serviços Web
JavaScript

Roberto Rocha

robertorocha@estgv.ipv.pt

Steven Abrantes

Steven@estgv.ipv.pt

JavaScript

- ***JavaScript*** é uma linguagem de programação interpretada
- Linguagem para programação *client-side* - ou seja, em contextos normais de desenvolvimento web, é interpretada pelo cliente (web browser) e não pelo servidor.
  - Exceção para programação server-side com javascript como Node.Js.
  - Isto irá permitir:
    - Alterar conteúdo da página (adicionar, remover, alterar, etc..);
    - Alterar estilos;
    - Efetuar validações;
    - Executar lógica sem a necessidade de efetuar pedidos ao servidor;
    - Efetuar pedidos Ajax;
    - Entre outros..

Ver slides em anexo

JavaScript - Declaração

- Código JavaScript pode ser declarado diretamente na página (tanto no Head como no Body) entre as tags html <script> e </script>. Ou pode ser declarado num ficheiro JavaScript externo (.js) e referenciado nas páginas HTML da seguinte forma:
  - <script src="<ficheiroJS>"></script>.

```
<script>
function myFunction() {
    document.getElementById("demo").inner
}
</script>
```

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

JavaScript - Sintaxe

- JavaScript é case-sensitive – por isso var1 é diferente de Var1.
- As variáveis JavaScript são declaradas com a keyword "var" `var variavel1;`
- As variáveis não são definidas por tipo, ou seja, até a sua definição, não têm tipo (não são string ou inteiro ou qualquer outro tipo de dados;

```
> var var1;
< undefined
> (typeof var1) //código para obter o tipo de variável
< "undefined"
> var1 = 0;
< 0
> (typeof var1) //código para obter o tipo de variável
< "number"
> var1 = "texto";
< "texto"
> (typeof var1) //código para obter o tipo de variável
< "string"
```

JavaScript - Sintaxe

- As variáveis têm de ter um identificador único e devem seguir as seguintes regras:
  - Nome da variável pode conter letras, números, "$" e "_";
  - Nome da variável tem de começar com uma letra, "$" ou "_";
  - Nomes das variáveis são case-sensitive;
  - Existem palavras reservadas em JavaScript que não podem ser utilizadas como nome de variáveis.

JavaScript - Sintaxe

- Podemos definir comentários em JavaScript da seguinte forma:
  - Entre "//" e o fim dessa mesma linha de código;
  - Entre "/*" (início) e "*/" (fim) - funciona para linhas múltiplas;
  - JavaScript reconhece também o início de comentários HTML "<!--" mas não interpreta como início de bloco, mas sim como interpreta o "//";
  - JavaScript não reconhece o final de comentários HTML "-->".

```
// Change heading:
document.getElementById("myH")
// Change paragraph:
document.getElementById("myP")
```

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML =
document.getElementById("myP").innerHTML =
```

JavaScript - Operadores

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

JavaScript (head, body ou externo)

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>

<body>

<h1>My Web Page</h1>

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

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

</body>
</html>
```

JavaScript (head, body ou externo)

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

<h1>My Web Page</h1>

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

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

<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

JavaScript (head, body ou externo)

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

JavaScript (output)

JavaScript can "display" data in different ways:

- Writing into an alert box, using **window.alert()**.
- Writing into the HTML output using **document.write()**.
- Writing into an HTML element, using **innerHTML**.
- Writing into the browser console, using **console.log()**.

JavaScript (output)

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

JavaScript (output)

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

JavaScript (output)

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

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

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

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

JavaScript (output)

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

JavaScript - Operadores

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

JavaScript – Tipo de variáveis

- Podemos ter diferentes tipos de variáveis em JavaScript sem para isso ser necessário definirmos o tipo na declaração.

```
var length = 16;                                    // Number
var lastName = "Johnson";                           // String
var cars = ["Saab", "Volvo", "BMW"];                // Array
var x = {firstName:"John", lastName:"Doe"};         // Object
```

JavaScript - Funções

- Uma função em JavaScript é definida pela keyword "function", seguido do nome da função e "()".
- Dentro dos parentesis podem estar definidos parametros da função.
- O nome das funções segue as mesmas regras das variáveis.
- O código das funções tem de ser definido entre "{" e "}".

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

JavaScript - Funções

```
function myFunction(p1, p2) {
    return p1 * p2;
}
```

```
var x = myFunction(4, 3);

function myFunction(a, b) {
    return a * b;
}
```

```
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

JavaScript - Funções

- JavaScript pode "apresentar" ao utilizador informação de diferentes formas:
  - Em janela de alerta: window.alert("texto");
  - Diretamente no HTML: document.write("texto");
  - Dentro de um elemento HTML: $(elemento).innerHTML = "texto";
  - Na consola do browser: console.log("texto");

JavaScript - Funções

- As funções JavaScript podem ser invocadas da seguinte forma:
  - Quando ocorre um evento;
  - Quando é invocada diretamente no código JavaScript;
  - Automáticamente (auto invocada). Ex.: (function(){…])();
- As funções JavaScript podem ser terminadas quando chegam a um *return,* podendo da mesma forma retornar valores.
- As funções em JavaScript são muito importantes porque permitem reutilização de código.

Ver slides em anexo

JavaScript – Arrays e Objetos

- Tal como em outras linguagens, JavaScript permite usar Arrays para armazenar múltiplos valores numa única variável.
- Criação de um Array:

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

- Obter um elemento de um Array:

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

- Alterar um elemento de um Array:

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

JavaScript – Arrays e Objetos

- Um Array em JavaScript é um tipo especial de um Objeto:
  - Um Array utiliza o índice para aceder aos valores;
  - Um outro objeto utiliza nomes para obter os valores do Objeto.
- Declaração do Objeto:

```
var person = {firstName:"John", lastName:"Doe", age:46};
```

- Obtenção de valores:

```
person.firstName
```

- Nota: Podemos Objetos dentro de um Array e vice-versa.

JavaScript – Arrays e Objetos

- Métodos e propriedades dos Arrays:
  - Obter tamanho do Array: array.length
  - Ordenar o Array: array.sort();
  - Ordenar de forma inversa o Array: array.reverse();
  - Obter elementos em String: array.toString();
  - Adicionar elemento a Array: array.push(elemento);
  - Remover o último elemento de um Array: array.pop();
  - Remover o primeiro elemento de um Array: array.shift();
  - Adicionar elemento ao início do Array: array.unshift(elemento);

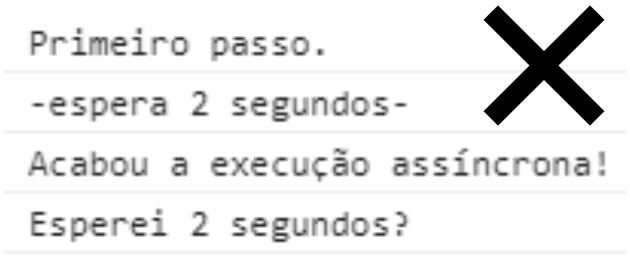
JavaScript - Pedidos Assíncronos

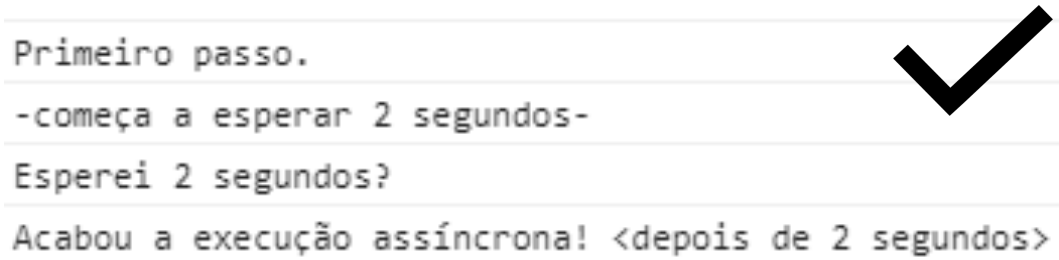
- Diferença entre pedidos síncronos e assíncronos:
  - Síncronos: Um pedido síncrono para a execução do restante código até o pedido concluir. Nesses casos, o motor JavaScript do browser está bloqueado.
  - Assíncronos: Um pedido assíncrono não bloqueia o cliente JavaScript. Nesse momento, o utilizador pode continuar a executar outras operações.

JavaScript - Pedidos Assíncronos

- A função setTimeout() é um exemplo de uma chamada assíncrona.
- Vamos ver como funciona:

```
// primeiro passo
console.log("Primeiro passo.");
// execução assíncrona
setTimeout(function() {
  console.log("Acabou a execução assíncrona!");
}, 2000);
// e isto vai sair quando?
console.log("Esperei 2 segundos?");
```





JavaScript - Pedidos Assíncronos (AJAX)

- AJAX = Asynchronous JavaScript And XML. - Não é uma linguagem de programação
- Faz uso do objeto do browser "XMLHttpRequest".
- Ajax permite a troca de informação com o servidor de forma assíncrona.
- AJAX é muito útil no desenvolvimento porque permite:
  - Obter dados de um servidor mesmo após a página ser carregada;
  - Atualizar página ou partes da página sem a recarregar;
  - Enviar dados ao servidor em background e de forma assíncrona.
- Os pedidos devem ser feitos ao mesmo servidor que contém a página, porque os browsers atuais não permitem Access Across Domains.

JavaScript - Pedidos Assíncronos (AJAX)

- Criar um pedido AJAX:

```
var xhttp = new XMLHttpRequest();
```

- Métodos do XMLHttpRequest:

new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(*method, url, async, user, psw*)	Specifies the request *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

JavaScript - Pedidos Assíncronos (AJAX)

- Propriedades do XMLHttpRequest:

onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

JavaScript - Pedidos Assíncronos (AJAX)

- Efetuar um pedido ao servidor:

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

    - O 3º parâmetro define se o pedido é síncrono (false) ou assíncrono (true).

- GET vs. POST:
    - GET é mais simples e mais rápido que o POST e pode ser utilizado quase sempre;
    - POST deve ser utilizado quando:
        - Queremos enviar um ficheiro para o servidor;
        - Estamos a enviar uma grande quantidade de dados para o servidor;
        - Enviamos dados introduzidos por um utilizador – POST é mais robusto que GET, no controlo de valores.

JavaScript - Pedidos Assíncronos (AJAX)

```
xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", true);
xhttp.send();


xhttp.open("POST", "ajax_test.asp", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Henry&lname=Ford");
```

JavaScript - Callbacks

- As funções de Callback permitem definir código a ser executado quando o pedido efetuado é concluído.
  - No caso de setTimeout, temos: *setTimeout(<callback>, <delay>);*
  - Conjugando com um pedido Ajax, temos:

```
var xhttp;
xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
  callbackFunction(this);
}
};
xhttp.open("GET", url, true);
xhttp.send();
```

- Uma função de Callback, terá de ser passada como parâmetro para outras funções assíncronas (que o permitam).

JavaScript - Promises

- Promises em JavaScript são objetos que encapsulam uma operação assíncrona e notifica quando a execução é concluída.
- Pode ser utilizado de uma forma semelhante às Callbacks para execução de código após a conclusão da execução, no entanto, providencia dois métodos, um para quando a execução foi concluída com sucesso (then...) e outra para quando ocorreu um erro (catch...).

```
someAsyncOperation(someParams)
.then(function(result){
    // Do something with the result
})
.catch(function(error){
    // Handle error
});
```

JavaScript - Promises

- Não confundir com Try, catch. Até porque não vai considerar erro se for tratado anteriormente com o Promise.

```
try{
    axios.get('http://www.somepage.com')
    .then(function response {
        // Handle response
    })
} catch (error){
    // We will never end up here, even if there is an error thrown
inside the Promise chain
}
```

JavaScript - Promises

- Criar Promise:

```
var promise = new Promise(function(resolve, reject) {
  const x = "geeksforgeeks";
  const y = "geeksforgeeks"
  if(x === y) {
    resolve();
  } else {
    reject();
  }
});

promise.
    then(function () {
        console.log('Success, You are a GEEK');
    }).
    catch(function () {
        console.log('Some error has occured');
    });
```

Output:

```
Success, You are a GEEK
```

JavaScript - Promises

```
var promise = new Promise(function(resolve, reject) {
    reject('Promise Rejected')
})

promise
    .then(function(successMessage) {
        console.log(successMessage);
    })
    .catch(function(errorMessage) {
       //error handler function is invoked
        console.log(errorMessage);
    });
```

JavaScript – EventListener

```
element.addEventListener(event, function, useCapture);
```

```
element.addEventListener("click", function(){ alert("Hello World!"); });
```

JavaScript – EventListener

```
element.addEventListener("click", myFunction);

function myFunction() {
  alert ("Hello World!");
}
```

JavaScript – EventListener

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

JavaScript – Manipulação da DOM

- DOM = **D**ocument **O**bject **M**odel. É a árvore de elementos de uma página.
- Podemos utilizar JavaScript para:
  - Alterar qualquer elemento ou atributo HTML na página;
  - Alterar qualquer estilo CSS da página;
  - Remover ou criar qualquer elemento ou atributo HTML na página;
  - Reagir a qualquer evento HTML da página;
  - Criar qualquer evento HTML na página.

JavaScript – Manipulação da DOM

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

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
```

document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element
element.setAttribute(*attribute, value*)	Change the attribute value of an HTML element

Ver mais https://www.w3schools.com/js/js_htmldom_document.asp

JavaScript – Validação de dados

- JavaScript pode ser utilizado também para efetuar a validação de formulários e dados introduzidos pelo utilizador.
- Estas validações têm a particularidade de permitir verificar os dados introduzidos pelo utilizador sem existir a necessidade de um pedido ao servidor.

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

JavaScript – Validação de dados

- Podemos utilizer JavaScript para validar os nossos dados em termos de:
  - Preenchimento;
  - Comprimento dos dados;
  - Tipo de dados;
  - Formato;
  - Valores repetidos;
  - Etc…

JavaScript – Validação de dados

- Adicionalmente, HTML5 fornece-nos um novo conceito chamado "constraint validation", que nos permite validar por:
  - Atributos do input HTML
  - Pseudo-selectors de CSS
  - Métodos e Propriedades da DOM

```
<form>
  <label for="choose">Would you prefer a banana or a cherry?</label>
  <input id="choose" name="i_like" required pattern="banana|cherry">
  <button>Submit</button>
</form>
```

```
input:invalid {
  border: 2px dashed red;
}

input:valid {
  border: 2px solid black;
}
```

```
var email = document.getElementById("mail");

email.addEventListener("input", function (event) {
  if (email.validity.typeMismatch) {
    email.setCustomValidity("I expect an e-mail, darling!");
  } else {
    email.setCustomValidity("");
  }
});
```

Ver mais https://www.w3schools.com/js/js_validation.asp

JavaScript – Armazenamento de dados

- Podemos utilizar JavaScript para armazenar dados em armazenamento local (no browser). Podemos por exemplo armazenar os dados como:
  - Cookies;
  - LocalStorage;
  - SessionStorage.

```
document.cookie = "username=John Doe";
localStorage.setItem("lastname", "Smith");
sessionStorage.setItem("lastname", "Smith");
```

- Cookies - Armazenamento de dados no computador local em ficheiros de texto pequenos;
- LocalStorage – Armazena pares chave-valor, sem data de expiração no browser;
- SessionStorage – Armazena pares chave-valor no browser durante a sessão do utilizador.

JavaScript – Armazenamento de dados

- Cookies
  - Definimos uma cookie em texto:

```
document.cookie = "cookiename=cookievalue"
```

  - Podemos definir uma data de expiração:

```
document.cookie = "cookiename=cookievalue; expires= Thu, 21 Aug 2014 20:00:00 UTC"
```

  - E até um caminho para organizar em pastas:

```
document.cookie = "cookiename=cookievalue; expires= Thu, 21 Aug 2014 20:00:00 UTC; path=/ "
```

  - Obter uma Cookie:

```
var x =  document.cookie
```

  - Eliminar uma Cookie – Devemos definir o conteúdo como vazio e a data de expiração anterior à data atual:

```
document.cookie = "cookiename= ; expires = Thu, 01 Jan 1970 00:00:00 GMT"
```

JavaScript – Armazenamento de dados

- LocalStorage
  - Definir um elemento chave-valor:

```
window.localStorage.setItem('melhor Disciplina', 'Programação e Serviços Web');
```

  - Obter um elemento:

```
window.localStorage.getItem('Programação e Serviços Web');
```

  - Remover um elemento:

```
window.localStorage.removeItem('melhor Disciplina');
```

  - Limpar o armazenamento:

```
window.localStorage.clear()
```

  - Obter valor da chave por index:

```
window.localStorage.key(0)
```

JavaScript – Armazenamento de dados

- LocalStorage – Características
  - Não é seguro usar localStorage para guardar dados sensíveis;
  - Em qualquer browser, o localStorage é limitado a guardar apenas 5Mb de dados. (Ainda assim, no caso das Cookies é 4Kb);
  - O uso do localStorage é síncrono;
  - LocalStorage só armazena String, se quisermos armazenar objetos terão de ser manipulados (JSON string por exemplo);
  - Não pode ser usado por web workers.

JavaScript – Armazenamento de dados

- SessionStorage
  - Igual à LocalStorage, mas os dados apenas são guardados durante uma sessão.
  - Uma sessão dura enquanto o browser estiver aberto e mantem-se durante atualizações de página.
  - Uma sessão é criada sempre que abrimos uma nova janela/tab no browser, e é eliminada (e limpar o SessionStorage) sempre que fecharmos uma janela/tab.
  - A sintaxe é igual à Local Storage.

JavaScript - Anexos

JavaScript

O JavaScript pode alterar os conteudos do HTML

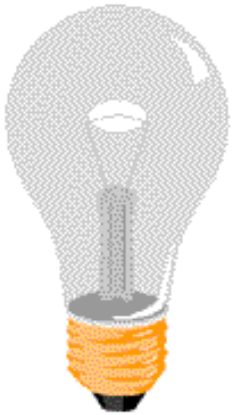
getElementById()

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

<h1>What Can JavaScript Do?</h1>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button"
onclick="document.getElementById('demo').innerHTML = 'Hello
JavaScript!'">
Click Me!</button>

</body>
</html>
```

JavaScript

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

<h1>JavaScript Can Change Images</h1>

<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>

<script>
function changeImage() {
    var image = document.getElementById('myImage');
    if (image.src.match("bulbon")) {
        image.src = "pic_bulboff.gif";
    } else {
        image.src = "pic_bulbon.gif";
    }
}
</script>

</body>
</html>
```

JavaScript Can Change Images



Click the light bulb to turn on/off the light.

JavaScript

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

<h1>What Can JavaScript Do?</h1>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<script>
function myFunction() {
    var x = document.getElementById("demo");
    x.style.fontSize = "25px";
    x.style.color = "red";
}
</script>

<button type="button" onclick="myFunction()">Click Me!</button>

</body>
</html>
```

JavaScript

```
<input id="numb" type="number">

<button type="button" onclick="myFunction()">Submit</button>

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

<script>
function myFunction() {
    var x, text;

    // Get the value of the input field with id="numb"
    x = document.getElementById("numb").value;

    // If x is Not a Number or less than one or greater than 10
    if (isNaN(x) || x < 1 || x > 10) {
        text = "Input not valid";
    } else {
        text = "Input OK";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>
```

JavaScript (Metodos de Strings)

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

<p id="p1">Please locate where 'locate' occurs!.</p>

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

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

<script>
function myFunction() {
    var str = document.getElementById("p1").innerHTML;
    var pos = str.indexOf("where");
    document.getElementById("demo").innerHTML = pos;
}
</script>

</body>
</html>
```

Please locate where 'locate' occurs!.

Try it

14

JavaScript (Metodos de Strings)

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7,13);
```

Banana

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft","W3Schools");
```

Please visit W3Schools!

```
var text1 = "Hello World!";
var text2 = text1.toUpperCase();
```

HELLO WORLD!

```
var text1 = "Hello";
var text2 = "World";
text3 = text1.concat(" ",text2);
```

Hello World!

```
var str = "HELLO WORLD";
str.charAt(0);            // returns H
```

JavaScript (Datas)

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

<script>
document.getElementById("demo").innerHTML = Date();
</script>


<script>
var d = new Date("October 13, 2014 11:13:00");
document.getElementById("demo").innerHTML = d;
</script>



var d = new Date("2015-03-25");
```

JavaScript (If...Else)

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

```
if (hour < 18) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

JavaScript (Switch)

```
switch(expression) {
    case n:
        code block
        break;
    case n:
        code block
        break;
    default:
        default code block
}
```

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

JavaScript (Loop)

```
for (statement 1; statement 2; statement 3) {
    code block to be executed
}
```

```
for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
}
```

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

```
while (i < 10) {
    text += "The number is " + i;
    i++;
}
```

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

```
do {
    text += "The number is " + i;
    i++;
}
while (i < 10);
```