

A linguagem JavaScript





Linguagem

- Se você lembra de C++, vai achar fácil
- Estruturas de comparação (if, switch) iguais
- Laços de repetição (for, while, do while) iguais
- Operadores (&&, ||, ==, !=, !, ++, --, +=, -=, etc.) iguais
- Comentários (//, /* ... */) iguais



Comentários

// Comentário de linha

/* Comentário
de
Bloco */

Variáveis

- Na declaração, usa-se **var** antes do nome da variável.
- Os tipos de dados são inferidos. Logo, não são declarados.

```
var a = 10; // int
```

```
var b = 1.99; // float
```

```
var c = true; // bool
```

```
var d = "Thiago"; // string
```

Constantes

- Coloca-se a palavra **const** na frente do nome.
- Assim como nas variáveis, os tipos de dados são inferidos.

```
const PI = 3.1416; // float
```

```
const TITULO = "Loja virtual"; // string
```

Conversão de tipos

- Feita automaticamente
- Use com atenção

```
var x = 10; // int
```

```
var y = "20"; // string
```

```
var z = x * y; // 200 (int)
```

```
var s = x + y; // "1020" (string)
```

Regra: Soma com string sempre resulta em string.



Conversão de tipos: string para int

- Use a função `parseInt`:

```
var x = parseInt( "100" );
```



Conversão de tipos: string para float

- Use a função `parseFloat`:

```
var x = parseFloat( "100.00" );
```


Conversão de tipos: para string

- Use o método toString()

// int para string

```
var x = 10;
```

```
var s1 = x.toString();
```

// float para string

```
var y = 10.01;
```

```
var s2 = y.toString();
```

Conversão de tipos: para string

- Também pode concatenar com uma string !

// int para string

```
var x = 10;
```

```
var s1 = x + "";
```

// float para string

```
var y = 10.01;
```

```
var s2 = y + "";
```

Arredondamento

- Use `Math.round`:

```
var x = Math.round( 1.9876 );  
// x = 2
```

- Para arredondar pra N casas decimais:

```
// Ex: 2 casas decimais  
var x = Math.round( 100 * 1.9876 ) / 100;  
// x = 1.99
```



Geração de número aleatório

- Use `Math.random`, que gera um valor de ponto flutuante entre 0 e 1.

```
var x = Math.random();  
// ex: x = 0.576893812
```

Arrays

```
var nomes = new Array( "Juca", "Bia",  
    "Mário", "Paula", "Sérgio" );
```

```
// Imprime Juca na página  
document.write( nomes[ 0 ] );  
// Imprime Sérgio na página  
document.write( nomes[ 4 ] );  
// Altera Bia para Carla  
nomes[ 1 ] = "Carla";
```



Arrays

```
// Numero de elementos
document.write( nomes.length );
// Adicionando um elemento
nomes.push( "Roberto" );
// Procurando "Roberto"
var pos = nomes.indexOf("Roberto");
// Removendo "Roberto" pela posição
nomes.splice( pos, 1); // 1 elemento
```

Operadores de Atribuição

- Como no C++ !

```
var i = 10;  
i += 5; // 15  
i *= 2; // 30  
i /= 3; // 10  
i -= 5; // 5  
i %= 2; // 1
```

```
// Operadores Aritméticos  
i++; // Incrementa em 1  
i--; // Decrementa em 1
```

Operadores de Comparação

- Idem ao C++ !
 - `a == b` // igual
 - `a != b` // diferente
 - `a > b` // maior
 - `a < b` // menor
 - `a >= b` // maior ou igual
 - `a <= b` // menor ou igual
- Porém, há o `===` que testa se os **tipos** e os **valores** são iguais.
 - `a === b` // valor e tipo iguais ?
- Similarmente, existe o `!==`.
 - `a !== b` // valor e tipo diferentes ?

Operadores Lógicos

- Igual ao C++ !

- a **&&** b // e
- a **||** b // ou
- **!** a // negação

Operador Ternário

- Igual ao C++ !
 - `var c = (a > b) ? "Maior" : "Menor ou igual";`
 - `var tratamento = ("M" == sexo) ? "Sr." : "Sra.";`



if

```
if ( a > b )  
    document.write( "a é maior" );  
else if ( a < b )  
    document.write( "b é maior" );  
else  
    document.write( "a e b são iguais." );
```

switch

```
switch ( opcao )  
{  
    case 1: sabor = "Uva"; break;  
    case 2: sabor = "Limão"; break;  
    case 3: sabor = "Maçã"; break;  
    default: sabor = "Morango";  
}
```



for

```
for ( i = 0; ( i < 100 ); i++)
```

```
{
```

```
    document.write( "Número: " + i + "<br />" );
```

```
}
```

while

```
var i = 0;
while ( i < 100 )
{
    document.write( "Número: " + i + "<br />" );
    i++;
}
```



do while

```
var i = 0;
```

```
do
```

```
{
```

```
    document.write( "Número: " + i + "<br />" );
```

```
    i++;
```

```
} while ( i < 100 );
```

for ... in

```
var nomes = new Array( "Juca", "Bia",  
    "Mário", "Paula", "Sérgio" );  
var j;
```

```
for ( j in nomes ) // j irá de 0 a 4  
{  
    document.write( nomes[ j ] + "<br />" );  
}
```


funções

- Não se declara o tipo de retorno
- Não se declara o tipo dos parâmetros
- Possui a palavra **function** na frente do nome

```
function soma(a, b)
{
    return a + b;
}
```

```
var x = soma( 10, 20 ); // 30
var s = soma( "Ei ", "você !" ); // "Ei você !"
```

Função anônima

- Recurso que permite definir uma função diretamente a um objeto ou evento

```
document.querySelector( '#form-busca' ).onsubmit = function() {  
    if( document.querySelector( '#q' ).value == '' ) {  
        document.querySelector( '#form-busca' ).style.background =  
                                                    'red';  
        return false;  
    }  
}
```

Expressões com funções

- Em JavaScript, uma função pode ser atribuída a uma variável usando uma expressão

```
var x = function(a, b) {return a * b;}  
var z = x(4, 3);
```

Function Hoisting

- JavaScript, por padrão, move todas as declarações para o topo do escopo corrente (elevant = hoisting)

```
myFunction(5);  
//funciona!!!  
function myFunction(y) {  
    return y * y;  
}
```

Auto-invocando funções

- Funções podem invocar a si mesmas, automaticamente, sem serem chamadas. Basta envolvê-la entre parênteses e pôr um () ao fim:

```
(function () {  
    var x = "Mundo cruel!";  
}) ();
```

Funções podem ser usadas como valores

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

```
var x = myFunction(4, 3) * 2;
```

Funções são “objetos”

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

```
var txt = myFunction.toString();
```

Objetos em Javascript

Object



Properties

`car.name = Fiat`

`car.model = 500`

`car.weight = 850kg`

`car.color = white`

Methods

`car.start()`

`car.drive()`

`car.brake()`

`car.stop()`

Objeto Carro

- Valores simples:

```
var car = "Fiat";
```

- Múltiplos atributos

```
var car = {type:"Fiat", model:"500", color:"white"};
```

valores devem ser escritos em pares **chave: valor**

- Objeto completo

```
var car = {  
  type:"Fiat",  
  model:"500",  
  color:"white",  
  ligar: function() {return "VRUM!!!";}   
};
```

Métodos String

- **replace(valAnt, valNovo)** – altera o conteúdo da 1ª ocorrência (padrão)

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

para mudar todos:

```
var n = str.replace(/Microsoft/g, "Cefet");
```

/g – global match

- **toUpperCase(), toLowerCase()** – para conversão
- **charAt(), charCodeAt(), str[]** - não-seguro
- **split()** – converte uma string em array

```
var txt = "a, b, c, d, e";  
txt.split(",") // ["a", "b", "c", "d", "e"]
```

Métodos String

- **indexOf()** – retorna a posição da primeira ocorrência do texto

```
var str = "Please locate where 'locate' occurs!.";
```

```
var pos = str.indexOf("locate");
```

- **lastIndexOf()** – retorna a última ocorrência
- **search()** – idêntico ao **indexOf()** mas aceita expressões regulares
- Particionando uma string
 - **slice(start, end)** – extrai a parte e retorna numa nova string
 - **substring(start, end)** = **slice()** **NÃO aceita índice negativo**
 - **substr(start, length)** – similar ao **slice()**, muda o parâmetro...

Métodos Numéricos

- `base.toExponential({exponente})`
- `numero.toFixed({numCasasDec})` \$\$\$
- `parseInt()`, `parseFloat()`
- `Math` object
 - `Math.random()`
 - `Math.min()`, `Math.max()`
 - `Math.round()`, `Math.ceil()` ^, `Math.floor()` v
 - `Math.E`, `Math.PI`, `Math.SQRT2`, `Math.LN2`



Métodos de Datas

- `getDate()` - dia do mês
- `getDay()` - dia da semana
- `getFullYear()` - ano yyyy
- `getHours()` - 0-23
- `getTime` - milisegundos desde 1/1/1970

Métodos para arrays

- `toString()` – converte numa string
`join(separador)` = `toString()` com separador escolhido
- `pop()` `push()`
- `shift()`, `unshift()`
- `delete`
- `splice()` – adicionar novos elementos

```
var fruits = ["Banana", "Laranja", "Maçã"];  
fruits.splice(2, 0, "Limão", "Kiwi");
```

 - 2 – posição onde começar a inserir novos elementos
 - 0 – quantos elementos remover
 - "Limão", "Kiwi" – novos elementos
- `sort()`, `reverse()`
- `concat()`

JavaScript no HTML



```
<script>  
<!--  
function ga(o,e){  
  if (document.getElementById(  
    a=o.id.substring(1)  
    p = "";  
    r = "";  
    g = e.target;  
    if (g) {  
      t = g.id;  
      g = g.parentNode;  
    }  
  }  
}
```

Colocando JavaScript no HTML

```
<script type="text/javascript">  
  // Aqui fica o script  
</script>
```

- Pode ser colocado no **head** ou no **body**.



Sobre a execução

- Se o código não estiver dentro de uma função, ele será executado automaticamente ao carregar a página.
- Dentro de uma função, obviamente só será executado chamando a mesma.

Escrevendo em uma página HTML

```
<!doctype html>
<html>
<head>
<script>
    document.write( "Este texto será exibido
    automaticamente na página HTML." );
</script>
</head>
<body>
</body>
</html>
```

Escrevendo com formatação HTML em uma página HTML

```
<html>
```

```
<body>
```

```
<script type="text/javascript" >  
    document.write( "<strong>Texto  
    em negrito.</strong>" );
```

```
</script>
```

```
</body>
```

```
</html>
```

Exibindo uma mensagem de alerta

- Use **alert**:

```
<script type="text/javascript" >  
    alert( "JavaScript em execução!" );  
</script>
```

Exibindo uma pergunta

- Use **confirm**:

```
<script type="text/javascript" >  
  var confirmou = confirm( "Está achando fácil?" );  
  if (confirmou) // confirmou terá true ou false  
    document.write( "Ótimo" );  
  else  
    document.write( "Qual a sua dúvida ?" );  
</script>
```

Solicitando um valor do usuário

- Use **prompt**:

```
<script type="text/javascript" >  
  // prompt( titulo, valor default )  
  var resposta = prompt(  
    "Qual o seu nome ?", "Bob Esponja" );  
  document.write(  
    "Seu nome é " + resposta );  
</script>
```

Links HTML com JavaScript

- Use **javascript:** mais o comando no **href** do link

<!-- Repare o uso de aspas simples no texto passado como parâmetro -->

Oi

Botões HTML com JavaScript

- Use **javascript:** num **evento** do botão

<!-- Repare o uso de aspas simples no texto passado como parâmetro -->

```
<input type="button"  
  onclick="javascript:alert('Oi')"  
  value="Oi" />
```


Criando e chamando funções

```
<html>
<head>
<script type="text/javascript" >
function exibirAlerta()
{
    alert( "Seu espaço em disco está no fim." );
}
</script>
</head>
<body>
<a href="javascript:exibirAlerta();" >Alerta</a>
<br />
<input type="button" onclick="javascript:exibirAlerta();"
    value="Alerta" />
</body>
</html>
```



Eventos

- Cada elemento HTML possui eventos que podem disparar uma função JavaScript
- Exemplos:
 - Ao clicar
 - Quando uma página ou imagem for carregada
 - Ao passar o mouse sobre um link ou imagem
 - Ao apertar uma tecla
 - Ao selecionar um campo de um formulário
 - Ao submeter um formulário



Significado de eventos comuns

on <code>click</code>	Ao clicar
on <code>change</code>	Ao modificar
on <code>load</code>	Ao carregar/entrar na página
on <code>unload</code>	Ao sair da página
on <code>blur</code>	Ao perder o foco
on <code>focus</code>	Ao ganhar o foco
on <code>mouseover</code>	Ao mouse passar sobre algo
on <code>mouseout</code>	Ao mouse sair de algo
on <code>submit</code>	Ao submeter dados (form.)
on <code>reset</code>	Ao <i>resetar</i> um formulário

Exemplo: onblur

```
<html>
<head>
<script >
  function informaTamanho(){
    var texto = document.meuform.nome.value;
    alert( texto + " possui " + texto.length + "
    caracteres." );
  }
</script>
</head>
<body>
<form name="meuform" >
  <input type="text" name="nome"
  onblur="informaTamanho()" />
</form>
</body>
</html>
```

Exemplo: onmouseover, onmouseout

```
<!-- Ao passar o mouse sobre a imagem, será  
      trocada imagem. Ao deixar a imagem, volta ao  
      normal -->
```

```
<img src='verde.png'  
      onmouseover="this.src='vermelho.png';"  
      onmouseout="this.src='verde.png';" />
```

```
<br />
```

```
<!-- Acrescenta exclamação ao passar o mouse  
      sobre o botão -->
```

```
<input type='text' value='Olá !'  
      onmouseover="this.value += '!' ;" />
```



window

- Objeto usado para comunicação com o usuário e manipulação da janela
- Representa a janela do navegador ou de um frame
- É criado automaticamente ao haver uma tag `<body>` ou `<frameset>`



O uso de *window* é opcional

- `window.alert("Oi")` ou só `alert("Oi")`
- `window.confirm("Quer ?")` ou só `confirm("Quer ?")`
- `window.document.write("Oi")` ou só `document.write("Oi")`
- etc.

Propriedades de *window* mais usadas

document	Documento da janela
location	Representa a URL da página
navigator	Contém informações sobre o navegador
name	Nome da janela
status	Conteúdo da barra de status
innerHeight	Altura interna
innerWidth	Comprimento interno
outerHeight	Altura externa
outerWidth	Comprimento externo

Métodos de *window* mais usados

alert	Mensagem de alerta
confirm	Pede confirmação ao usuário
prompt	Pede um valor ao usuário
back	Simula o botão Voltar do navegador
forward	Simula o botão Avançar do navegador
open	Abre uma nova janela
close	Fecha uma janela
print	Imprime o conteúdo da janela



document

- O objeto `document` representa a página atual
- Pertence ao objeto `window`
- Possui todos os elementos da página, os quais podem ser acessados pelo *nome* ou pela *posição dentro do array*:
 - `document.meuform`
 - `document.meuform.email`
 - `document.forms[0]`
 - `document.forms[0].elements[0]`

document

- Um meio bastante utilizado para acessar os elementos do documento é o acesso pelo **id** do elemento:

```
var email =  
    document.getElementById( 'email' );
```

document

- Possui propriedades adicionais
 - `backgroundColor` // Cor de fundo
 - `fgColor` // Cor do texto
 - `linkColor` // Cor dos links
 - `vlinkColor` // Cor dos links pressionados
 - ...
- Possui métodos
 - `write("<p>Um parágrafo</p>");`
 - `getSelection();` // Retorna o texto selecionado
 - ...

location

- Manipula a localização da página

```
<script type="text/javascript" >  
  var site = prompt( "Ir para:",  
    "http://www.google.com/" );  
  // Acessa o site  
  location.href = site;  
</script>
```

navigator

- Possui informações sobre o navegador

```
<script type="text/javascript" >
    var nome = navigator.appName;
    var plataforma = navigator.platform;
    var usaJava = navigator.javaEnabled() ?
        "Sim" : "Não";

    alert( "Navegador: " + nome +
        "\nPlataforma:" + plataforma +
        "\nUsa Java:" + usaJava );
</script>
```



String e Crítica de Dados

- Um dos objetos mais usados para crítica (validação) de dados de formulários é o String
- Os valores informados pelo usuário serão objetos String
- String possui propriedades e métodos úteis e de fácil utilização



String e Crítica de Dados

- A maioria dos campos dos formulários possui a propriedade **value**.
- Essa propriedade é uma String, que é um objeto que possui atributos e métodos úteis para validação.

Exemplo

```
function validar()
{
    var nome = document.getElementById( 'nome' );
    // length é o comprimento da string
    if ( nome.value.length < 1 )
    {
        alert( 'Por favor, informe o Nome.' );
        // Põe o foco no campo Nome
        nome.focus();
        return false;
    }
    return true;
}
```

No formulário

```
<form id='meuform'
      name='meuform' method='post'
      action='cadastrar.php'
      onsubmit='return validar();' />

...

</form>
```

Outro exemplo

```
function validar()
{
    var email = document.getElementById('email');
    ...
    // indexOf procura pelo texto informado a partir da
    // posição informada. Retorna -1 se não encontrou ou
    // a posição onde encontrou o texto (0 ou mais).
    if ( email.value.indexOf('@', 0) < 0 )
    {
        alert( 'Por favor, informe arroba no email.' );
        email.focus();
        return false;
    }
    return true;
}
```

Verificando se um valor é número

- Use a função **isNaN**, que é a contração de **is Not a Number** (não é um número).
- A função retorna true se a string passada **não** for um número. Ex.:

```
if ( isNaN( campoQuantidade.value ) )  
{  
    alert( 'Informe um número na quantidade.' );  
}
```

Obtendo valor de um select

```
// Obtém a posição selecionada do select
var posicao = form.campoSelect.selectedIndex;

// Obtém, da opção na posição selecionada,
// o valor
var valor =
    form.campoSelect.options[ posicao ].value;
```

JavaScript Validation API

- `checkValidity()`

- true – input com dados válidos

```
<input id="id1" type="number" min="100" max="300" />
<button onclick="myFunction()">OK</button>
<p id="demo"></p>
<script>
  function myFunction() {
    var inpObj = document.getElementById("id1");
    if(inpObj.checkValidity() == false) {
      document.getElementById("demo").innerHTML =
        inpObj.validationMessage;
    }
  }
</script>
```

- `setCustomValidity()` – crie sua mensagem de validação

Propriedades de validação

Propriedade	Descrição
customError	true → usando msg de validação customizada
patternMismatch	true → valor do elemento fora do padrão
rangeOverflow	true → valor acima do definido no atrib max
rangeUnderflow	true → valor abaixo do definido no atrib min
stepMismatch	true → valor inválido para o atrib step
tooLong	true → valor excede o limite do atrib maxLength
typeMismatch	true → valor inválido para o atrib type
valueMissing	true → sem valor para campo requerido (required)
valid	true → valor válido

Usando arquivos externos

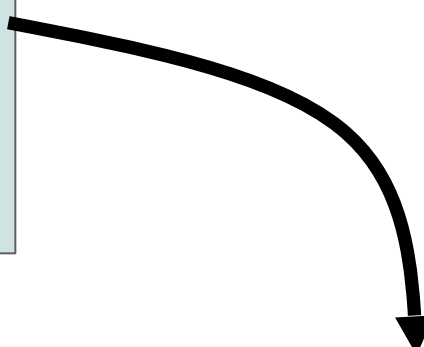
- É possível fazer uso de arquivos JavaScript (.js) externos.
- Basta que na tag `<script>` seja passado o arquivo na propriedade `src`.

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


Chamando função de arquivo externo

funcoes.js

```
function soma(a, b)
{
    return a + b;
}
```



teste.html

```
...
<script type="text/javascript" src="funcoes.js" >
    var x = soma( 10, 20 );
    document.write( "Resultado: " + x );
</script>
...
```



JSON

CEFET

Programação para Web 1

Prof. Dacy Câmara Lobosco



JSON

- **JSON** (JavaScript Object Notation - Notação de Objetos JavaScript) é uma formatação leve de troca de dados. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar. Está baseado em um subconjunto da linguagem de programação JavaScript, Standard ECMA-262 3a Edição -Dezembro - 1999.
- JSON é em formato texto e completamente independente de linguagem, pois usa convenções que são familiares às linguagens C e familiares, incluindo C++, C#, Java, JavaScript, Perl, Python e muitas outras. Estas propriedades fazem com que JSON seja um formato ideal de troca de dados.

JSON

```
{ "menu":  
  { "id": 31,  
    "texto": "Arquivo",  
    "itens": {  
      "menuitem": [  
        { "texto": "Novo", "onclick": "CriarNovoDoc()" },  
        { "texto": "Abrir", "onclick": "AbrirDoc()" },  
        { "texto": "Fechar", "onclick": "CloseDoc()" }  
      ]  
    }  
  }  
}
```

```
<menu id="31" texto="Arquivo">  
  <popup>  
    <menuitem value="New" onclick="CreateNewDoc()" />  
    <menuitem value="Open" onclick="OpenDoc()" />  
    <menuitem value="Close" onclick="CloseDoc()" />      </popup>  
</menu>
```



JSON

- A simplicidade de JSON tem resultado em seu uso difundido, especialmente como uma alternativa para XML em AJAX.
- Uma das vantagens reivindicadas de JSON sobre XML como um formato para intercâmbio de dados neste contexto, é o fato de ser muito mais fácil escrever um analisador JSON.
- Em JavaScript mesmo, JSON pode ser analisado trivialmente usando a função `eval()`.
- Isto foi importante para a aceitação de JSON dentro da comunidade AJAX devido a presença deste recurso de JavaScript em todos os navegadores web atuais.

Programação Funcional

- Assim como POO, programação funcional é uma forma de se pensar em como resolver problemas.
- Criada por Turing e Church na década de 1930, ficou restrita ao meu acadêmico... até agora!

$$e ::= x \mid \lambda x:\tau.e \mid e e$$
$$e ::= x \mid (x) \rightarrow \text{ifft}(\tau, x); e \mid e e$$



Conceitos de PF

- Em POO, a menor parte de um sistema é um objeto (atributos + métodos)
- Em PF, a menor parte de um sistema é uma função
 - Podemos atribuir funções a variáveis, passá-las como parâmetro, fazer com que uma função retorne uma outra função
 - Outras linguagens
 - Imutabilidade – todo valor é tratado como se fosse constante

High Order Functions

- Uma função que recebe outra função como parâmetro ou devolve uma função como resultado

Exemplo:

click é uma high order function

```
$( "btOk" ).click( function( )  
    {  
        alert( "Press!!!" );  
    } ) ;
```

função anônima é um callback



Função Callback

- Função de retorno. É uma função passada como argumento de outra função e é acionada dentro da função high order.
- É um importante recurso para lidar com manipulações assíncronas.



Escopo

- Variáveis globais podem ser vistas por todos, variáveis locais têm visibilidade limitada

Closures – “clausuras”

- Uma função guarda as variáveis do contexto em que foi criada:

```
function counter() {  
  var x = 0;  
  return function() {  
    return ++x;  
  }  
}  
  
var count = counter();  
console.log(count());  
console.log(count());  
console.log(count());  
console.log(x); //erro
```

Currying

- Transformando uma função com vários parâmetros em uma série de chamadas de funções com apenas um parâmetro cada

- Evitar que um parâmetro seja passado toda hora:

```
function hey(texto, nome) {  
    console.log(texto + ", " + nome);  
}  
  
hey("Bom dia", "João");  
hey("Bom dia", "José");  
hey("Bom dia", "Nicolau");
```

Reescrevendo

```
function hey(texto) {  
    return function(nome) {  
        console.log(texto + ", " + nome);  
    }  
}  
  
var bomDia = hey("Bom dia");  
bomDia("João");  
bomDia("José");  
bomDia("Nicolau");
```



jQuery

Biblioteca JavaScript cross-browser

jQuery is a lightweight, “write less, do more”, JavaScript library

w3schools.com/jquery



JQuery

- O Jquery é uma biblioteca criada para simplificar scripts client-side que interagem com HTML
 - Lançada em 2006
 - Resolve os problemas:
 - Incompatibilidade entre navegadores
 - Reduz código
 - Promove a reutilização de código através de plugins
 - Componentização através de plugins
 - Trabalha com AJAX e DOM



Simplificar código JavaScript

JavaScript puro:

```
document.getElementById("teste").value = 5;
```

Mesmo código em JQuery:

```
$("teste").val(5);
```


Como usar JQuery

- Faça o download da versão atual estável em jquery.com
- Acrescente o arquivo jquery.js na sua página.

```
<script src="jquery.min.js"></script>
```

o atributo `type="text/javascript"` não é mais necessário em HTML5



Sintaxe básica jQuery

- A maior parte do uso do JQuery trata a manipulação de elementos HTML ao executar algum evento
 - Eventos
 - Efeitos
 - Manipulação de DOM
 - AJAX
 - Plugins

Seletores JQuery

- `$(seletor).metodo()`
 - `$` - define o acesso ao objeto jQuery
 - `seletor` - elemento a ser manipulado
 - `metodo()` - método/ação a ser executado
 - Exemplos:

```
$(this).hide(); //elemento atual
```

```
$("p").hide(); //todos os parágrafos da página
```

```
$(".test").hide(); //classe test
```

```
$("#test").hide(); //id=test
```

- **Os seletores são os mesmos do CSS**

Document Ready Event

- Para que os métodos jQuery executem apenas quando a página estiver carregada (ready), coloque seu código dentro do evento document ready event:

```
$(document).ready(function(){  
    // seu código aqui  
});
```

- Seu uso é tão comum que foi simplificado para:

```
$(function(){  
    // seu código aqui  
});
```

Eventos JQuery

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

- Um evento representa o momento preciso quando alguma coisa acontece
- Sintaxe para eventos:
 - Para a maioria dos eventos DOM tem um evento jquery equivalente:
`$("p").click(//fazer algo aqui!!!);`

O método on()

- Organize seu código com o método on(). Assim, você poderá listar todos os eventos de um elemento:

```
$("#p").on("click", function({  
    alert("callback");  
}));
```

```
$("#p").on({  
    mouseenter: function() {  
        $(this).css("background-color", "lightgray");  
    },  
    mouseleave: function() {  
        $(this).css("background-color", "lightblue");  
    },  
    click: function() {  
        $(this).css("background-color", "yellow");  
    }  
});
```

Method	Description
<u>animate()</u>	Runs a custom animation on the selected elements
<u>clearQueue()</u>	Removes all remaining queued functions from the selected elements
<u>delay()</u>	Sets a delay for all queued functions on the selected elements
<u>dequeue()</u>	Removes the next function from the queue, and then executes the function
<u>fadeIn()</u>	Fades in the selected elements
<u>fadeOut()</u>	Fades out the selected elements
<u>fadeTo()</u>	Fades in/out the selected elements to a given opacity
<u>fadeToggle()</u>	Toggles between the <code>fadeIn()</code> and <code>fadeOut()</code> methods
<u>finish()</u>	Stops, removes and completes all queued animations for the selected elements
<u>hide()</u>	Hides the selected elements
<u>queue()</u>	Shows the queued functions on the selected elements
<u>show()</u>	Shows the selected elements
<u>slideDown()</u>	Slides-down (shows) the selected elements
<u>slideToggle()</u>	Toggles between the <code>slideUp()</code> and <code>slideDown()</code> methods
<u>slideUp()</u>	Slides-up (hides) the selected elements
<u>stop()</u>	Stops the currently running animation for the selected elements
<u>toggle()</u>	Toggles between the <code>hide()</code> and <code>show()</code> methods

Efeitos Hide & Show, Toggle

- Sintaxe:

```
$(seletor).hide(speed*, callback*);
```

```
$(seletor).show(speed*, callback*);
```

```
$(seletor).toggle(speed*, callback*);
```

speed em milisegundos ou ["slow"|"fast"]

*opcionais



Efeito fade

- fadeIn
- fadeOut
- fadeToggle

```
$(seletor).fadeIn(speed*, callback*);
```

```
$(seletor).fadeOut(speed*, callback*);
```

```
$(seletor).fadeToggle(speed*, callback*);
```

- fadeTo

```
$(seletor).fadeTo(speed*, opacity*, callback*);
```



Efeito Slide

- slideDown
- slideUp
- slideToggle

```
$(seletor).slideDown(speed*, callback*);
```

```
$(seletor).slideUp(speed*, callback*);
```

```
$(seletor).slideToggle(speed*, callback*);
```

Efeito Animation

```
$(selector).animation({params}, speed, callback);
```

- Exemplo:

```
$("#button").click(function(){  
    $("#div").animate({  
        left: '250px',  
        opacity: '0.5',  
        height: '150px',  
        width: '150px'  
    });  
});
```

```
$("#button").click(function(){  
    $("#div").animate({  
        height: 'toggle' //hide ou show  
    });  
});
```

Efeito stop

- Parar um efeito antes do seu término previsto

```
$(seletor).stop(stopAll, goToEnd);
```

- *stopAll* – se houver uma fila de efeitos, para todos. Por default, é false.
- *goToEnd* – encerra a animação mas leva ao fim do efeito. default = false;

```
$("#start").click(function(){  
    $("#div").animate({left: '100px'}, 5000);  
    $("#div").animate({fontSize: '3em'}, 5000);  
});
```

```
$("#stop").click(function(){  
    $("#div").stop();  
});
```

```
$("#stop2").click(function(){  
    $("#div").stop(true);  
});
```

```
$("#stop3").click(function(){  
    $("#div").stop(true, true);  
});
```

Manipulação de DOM (Document Object Model)

- Obtendo conteúdo
 - `text()` – texto de um elemento selecionado
 - `html()` – conteúdo do elemento, incluindo marcações de tags filhas
 - `val()` – valor de campos de formulário

```
$(seletor).text();
```

```
$(seletor).html();
```

```
$(seletor).val();
```

- `attr(nomeAtrib)` – valor de um atributo

```
$(seletor).attr(nomeAtrib);
```

Alterando conteúdo

- Obtendo conteúdo
 - text(**val**) – onde **val** é somente texto
 - html(**val**) – onde **val** pode ser texto e tags
 - val(**val**) – onde **val** deve ser compatível com o type do elemento de formulário

```
$(seletor).text("valor exemplo");
```

```
$(seletor).html("<h1>valor exemplo</h1>");
```

```
$(seletor).val("valor exemplo");
```

- attr(nomeAtrib, val) – valor de um atributo

```
$(seletor).attr(nomeAtrib, "valor atrib");
```

exemplo:

```
$("#w3s").attr({  
    "href" : "http://www.w3schools.com/jquery",  
    "title" : "W3Schools jQuery Tutorial"  
});
```