

# Programação Web

- ✓ Resolução do exercício 2 da aula anterior
- ✓ Formulário e CSS
- ✓ JavaScript
- ✓ Formas de utilização
- ✓ Conceitos básicos
- ✓ Tipos de variáveis
- ✓ Comentários
- ✓ Operadores (Aritméticos e Atribuição)
- ✓ Strings
- ✓ Caixas de Diálogo
- ✓ Conversão de tipos
- ✓ Ferramenta Desenvolvedor
- ✓ Exemplos
- ✓ Exercícios

Estrutura básica do HTML.  
1-Utilização das tags (sintaxe)  
2-Tags básicas.

## Exemplo Básico de HTML

Linguagem	Função
HTML	Estrutura da página, texto.
CSS	Formatação do visual da página, site.
JavaScript	Programação, interação, funcionalidades.

O que preciso saber até o momento:

- Entender os comandos (tags) do HTML
- Saber trabalhar com CSS básico (classe e id)
- Saber para que usar HTML e CSS

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Título do documento</title>
</head>
<body>
  <p><b>Exemplo Básico de HTML</b></p>
  <table border="1">
    <tr>
      <td><b>Linguagem</b></td>
      <td><b>Função</b></td>
    </tr>
    <tr>
      <td>HTML</td>
      <td>Estrutura da página, texto.</td>
    </tr>
    <tr>
      <td>CSS</td>
      <td>Formatação do visual da página, site.</td>
    </tr>
    <tr>
      <td>JavaScript</td>
      <td>Programação, interação, funcionalidades.</td>
    </tr>
  </table>
  <div>
    O que preciso saber até o momento:<br>
    <ul>
      <li>Entender os comandos (tags) do HTML</li>
      <li>Saber trabalhar com CSS básico (classe e id)</li>
      <li>Saber para que usar HTML e CSS</li>
    </ul>
  </div>
</body>
</html>
```

```
<head>
<meta charset="utf-8">
<title>Olimpíadas 2012</title>

<link rel="stylesheet" href="estilo.css" type="text/css" />

</head>

<body>

<table border="1" class="sombra">
  <tr>
    <td>&nbsp;&nbsp;&nbsp;Conteúdo&nbsp;&nbsp;&nbsp;</td>
    <td>
      <ul>
        <li>&nbsp;&nbsp;&nbsp;Esportes&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</li>
        <li>&nbsp;&nbsp;&nbsp;Medalhas&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</li>
        <li>&nbsp;&nbsp;&nbsp;Países&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</li>
      </ul>
    </td>
  </tr>
</table>
```

**Estilo externo**



estilo .sombra definido  
no arquivo estilo.css

Como inserir o CSS externo

Aqui teremos:

**arquivo.CSS** e **arquivo.HTML**

Para usar o CSS no HTML, usamos a tag link no cabeçalho (head).

Conteúdo

- Esportes
- Medalhas
- Países





```
<head>
<meta charset="utf-8">
<title>Olimpíadas 2012</title>
```

```
<style type="text/css">
  table {
    box-shadow: 15px 15px 55px gray;
    border-radius: 5px;
  }
</style>
```

**Estilo incorporado**

```
</head>
```

```
<body>
```

```
<table border="1" >
  <tr>
    <td>&nbsp;&nbsp;&nbsp;Conteúdo&nbsp;&nbsp;&nbsp;</td>
    <td>
      <ul>
        <li>&nbsp;&nbsp;&nbsp;Esportes&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</li>
        <li>&nbsp;&nbsp;&nbsp;Medalhas&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</li>
        <li>&nbsp;&nbsp;&nbsp;Países&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</li>
      </ul>
    </td>
  </tr>
</table>
```

As tabelas usarão o estilo "default" antes definido para a tag table, quando não for especificado nenhum estilo na tag table. Nenhum argumento seria especificado.

Conteúdo	<ul style="list-style-type: none"><li>• Esportes</li><li>• Medalhas</li><li>• Países</li></ul>
----------	--

Quando necessário podemos configurar no CSS um estilo para uma TAG. Isso faz com que sempre que usarmos a TAG, ela fique com a mesma configuração.

## Formulário de Contato

Preencha todos os campos.

---

Nome :

Nome completo

Email :

E-mail Válido.

Assunto :

Mensagem :

Sua mensagem

Enviar

```
<form action="" method="post" class="modelo1">
  <h1>Formulário de Contato<br/>
    <span class="titulo2">Preencha todos os campos.</span>
</h1>
<div>
  <label class="rotulos">Nome :</label>
  <input id="nome" type="text" name="nome" placeholder="Nome completo" class="entrada" />
</div>
<div>
  <label class="rotulos">Email :</label>
  <input id="email" type="email" name="email" placeholder="E-mail Válido." class="entrada" />
</div>
<div>
  <label class="rotulos">Assunto :</label>
  <select name="opcoes" class="entrada">
    <option value=""></option>
    <option value="Dúvidas do produto">Dúvidas do produto</option>
    <option value="Atraso na entrega">Atraso na entrega</option>
  </select>
</div>
<div>
  <label class="rotulos">Mensagem :</label>
  <textarea id="mensagem" name="mensagem" placeholder="Sua mensagem" class="entrada"></textarea>
</div>
<div>
  <button type="button" class="botao">Enviar</button>
</div>
</form>
```



```
.modelo1 {
  width: 500px;
  background-color: #8b4513;
  padding: 20px 30px 20px 30px;
  font-size: 12px;
  font-family: verdana;
  color: #fff;
  text-shadow: 2px 2px 2px #000;
  border-radius: 25px;
  margin: 0 auto;
}

h1 {
  padding: 0 0 10px 0;
  border-bottom: 3px solid #000;
  text-align: center;
}

.titulo2 {
  font-size: 11px;
}
```

```
div {
  margin: 0 0 20px;
  display: flex;
}

.rotulos {
  width: 20%;
  text-align: right;
  padding-right: 10px;
  margin-top: 10px;
  font-weight: bold;
}

.entrada {
  border: none;
  color: #525252;
  height: 30px;
  line-height: 15px;
  padding: 5px 0 5px 5px;
  width: 70%;
  border-radius: 5px;
  box-shadow: 2px 2px 2px #000;
  background-color: #f5deb3;
}
```

```
#mensagem {
  height: 100px;
  padding: 5px 0 0 5px;
  width: 70%;
}

.botao {
  margin: auto;
  background-color: #000;
  border: none;
  padding: 10px 25px 10px 25px;
  color: #fff;
  border-radius: 4px;
  text-shadow: 1px 1px 1px #FFE477;
  font-weight: bold;
  box-shadow: 1px 1px 1px #3D3D3D;
}

.botao:hover {
  color: #333;
  background-color: #EBEBEB;
}
```

## CSS Padding

« Previous

---

The CSS padding properties define the space between the element border and the element content.

---

**Define o espaço entre a borda do elemento e o conteúdo do elemento.**

## CSS Margin

« Previous

---

The CSS margin properties define the space around elements.

---

**Define o espaço arredor dos elementos.**

Fonte: [http://www.w3schools.com/css/css\\_padding.asp](http://www.w3schools.com/css/css_padding.asp) e [http://www.w3schools.com/css/css\\_margin.asp](http://www.w3schools.com/css/css_margin.asp)

## Padding - Individual sides

In CSS, it is possible to specify different padding for different sides:

### Example

```
p {  
  padding-top: 25px;  
  padding-bottom: 25px;  
  padding-right: 50px;  
  padding-left: 50px;  
}
```

## Padding - Shorthand property

The padding property can have from one to four values.

- **padding: 25px 50px 75px 100px;**
  - top padding is 25px
  - right padding is 50px
  - bottom padding is 75px
  - left padding is 100px
- **padding: 25px 50px 75px;**
  - top padding is 25px
  - right and left paddings are 50px
  - bottom padding is 75px
- **padding: 25px 50px;**
  - top and bottom paddings are 25px
  - right and left paddings are 50px
- **padding: 25px;**
  - all four paddings are 25px

Fonte: [http://www.w3schools.com/css/css\\_padding.asp](http://www.w3schools.com/css/css_padding.asp)

## Margin - Individual sides

In CSS, it is possible to specify different margins for different sides of an element:

### Example

```
p {  
  margin-top: 100px;  
  margin-bottom: 100px;  
  margin-right: 150px;  
  margin-left: 50px;  
}
```

Fonte: [http://www.w3schools.com/css/css\\_margin.asp](http://www.w3schools.com/css/css_margin.asp)

## Margin - Shorthand property

The margin property can have from one to four values.

- **margin: 25px 50px 75px 100px;**
  - top margin is 25px
  - right margin is 50px
  - bottom margin is 75px
  - left margin is 100px
- **margin: 25px 50px 75px;**
  - top margin is 25px
  - right and left margins are 50px
  - bottom margin is 75px
- **margin: 25px 50px;**
  - top and bottom margins are 25px
  - right and left margins are 50px
- **margin: 25px;**
  - all four margins are 25px

## CSS Syntax

```
box-shadow: none|h-shadow v-shadow blur spread color |inset|initial|inherit;
```

## CSS Syntax

```
text-shadow: h-shadow v-shadow blur color|none|initial|inherit;
```

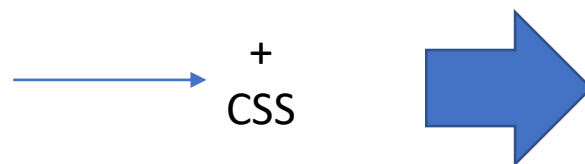
## Property Values

Value	Description
<i>h-shadow</i>	Required. The position of the horizontal shadow. Negative values are allowed
<i>v-shadow</i>	Required. The position of the vertical shadow. Negative values are allowed
<i>blur</i>	Optional. The blur distance
<i>color</i>	Optional. The color of the shadow. Look at <a href="#">CSS Color Values</a> for a complete list of possible color values

Fonte: <http://www.w3schools.com/>

mrc	m-	m+	÷
7	8	9	x
4	5	6	-
1	2	3	+
0	.	C	=

Form HTML



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Calculadora</title>
</head>
<body>
  <form>
    <div>
      <input type="text" name="res" id="res" value="">
    </div>
    <div>
      <button type="button">mrc</button>
      <button type="button">m-</button>
      <button type="button">m+</button>
      <button type="button">÷</button>
    </div>
    <div>
      <button type="button">7</button>
      <button type="button">8</button>
      <button type="button">9</button>
      <button type="button">x</button>
    </div>
    <div>
      <button type="button">4</button>
      <button type="button">5</button>
      <button type="button">6</button>
      <button type="button">-</button>
    </div>
```

```
      <div>
        <button type="button">1</button>
        <button type="button">2</button>
        <button type="button">3</button>
        <button type="button">+</button>
      </div>
      <div>
        <button type="button">0</button>
        <button type="button">.</button>
        <button type="button">C</button>
        <button type="button">=</button>
      </div>
    </form>
  </body>
</html>
```

Continua

```
<head>
  <meta charset="utf-8">
  <title>Calculadora</title>
  <link rel="stylesheet" href="style.css" type="text/css">
</head>
```

```
<form id="calculadora">
  <div class="visor">
    <input type="text" name="res" id="res" value="" class="visor_input">
  </div>
  <div class="calc_linhas">
    <button type="button" class="calc_botao_fun">mrc</button>
    <button type="button" class="calc_botao_fun">m-</button>
    <button type="button" class="calc_botao_fun">m+</button>
    <button type="button" class="calc_botao_op">÷</button>
  </div>
  <div class="calc_linhas">
    <button type="button" class="calc_botao">7</button>
    <button type="button" class="calc_botao">8</button>
    <button type="button" class="calc_botao">9</button>
    <button type="button" class="calc_botao_op">x</button>
  </div>
  <div class="calc_linhas">
```

Inserindo o CSS

```
    <div class="calc_linhas">
      <button type="button" class="calc_botao">0</button>
      <button type="button" class="calc_botao">.</button>
      <button type="button" class="calc_botao">C</button>
      <button type="button" class="calc_botao_res">=</button>
    </div>
  </form>
</body>
</html>
```



```
body {
  font-family: Verdana;
  font-size: 23px;
  color: #404040;
  background-color: #ffebed;
}

#calculadora {
  margin: 50px auto;
  height: 210px;
  padding: 15px;
  width: 190px;
  background-color: #3d4543;
  border: 1px solid #222;
  border-radius: 4px;
  box-shadow: 0 0 5px #000;
}

.visor {
  margin: 0 0 20px;
  padding: 3px;
  background-color: #222;
  border-radius: 3px;
}
```

```
.visor_input {
  display: block;
  width: 90%;
  height: 35px;
  padding: 0 8px;
  font-family: monospace;
  color: #444;
  text-align: right;
  background-color: #bccd95;
  border: 1px solid #222;
  border-radius: 2px;
  box-shadow: inset 0 2px 3px #000;
}

.calc_botao, .calc_botao_op, .calc_botao_fun, .calc_botao_res {
  float: left;
  padding: 0;
  margin: 0 0 5px 7px;
  width: 39px;
  line-height: 23px;
  color: white;
  text-align: center;
  text-decoration: none;
  text-shadow: 0 1px #000;
  background-color: #313131;
  border: 1px solid #000;
  border-radius: 3px;
  cursor: pointer;
  box-shadow: 1px 1px 1px #000;
}
```

```
.calc_botao:active {
  background-color: #282828;
}

.calc_botao_op {
  background-color: #ff4561;
}

.calc_botao_fun {
  background-color: #6f6f6f;
}

.calc_botao_res {
  background-color: #ffa70c;
}
```

style.css

observe a utilização de estilos "em cascata"

## O que é isso ?

```
const info = document.querySelector("#info");

const getLista = async (user) => {
  const response = await fetch(`https://api.github.com/users/${user}/repos`);
  const data = await response.json();
  info.innerHTML = `${data[3].id} - ${data[3].name}: ${data[3].html_url}`;
};

getLista("alcidestbj");
```



```
<html>
<head>
  <title>Exemplo JavaScript</title>
  <script>
    alert("Eu estou no cabeçalho.");
  </script>
</head>
<body>
  <script>
    alert("Eu estou no corpo do documento.");
  </script>
</body>
</html>
```

Adicionando o código direto no arquivo HTML.

Usando arquivos externos

```
<html>
<head>
  <title>Exemplo JavaScript</title>
  <script src="ex3.js"></script>
</head>
<body>
  Conteúdo da página.
  <script src="ex4.js"></script>
</body>
</html>
```

Uma única tag `<script>` não pode ter o atributo `src` e código interno juntos.

Ex errado:

```
<script src="exemplo1.js">  
    console.log("olá mundo");  
</script>
```

Ex **CORRETO:**

```
<script src="exemplo1.js"></script>  
  
<script>  
    console.log("olá mundo");  
</script>
```

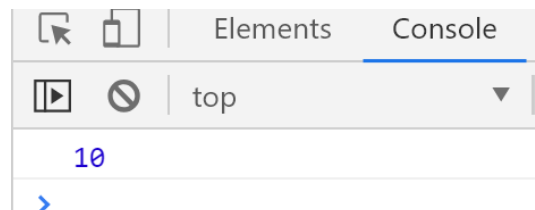
Como temos várias versões do JS, os códigos precisam manter a compatibilidade. Por este motivo, algumas coisas antigas ainda funcionam hoje.

Se quisermos usar o modo “**novo ou moderno**”, precisamos inserir a instrução “**use strict**”; na primeira linha em um primeiro momento.

Em muitos casos, quando usamos recursos modernos do ES6 ou superior, isso é automático.

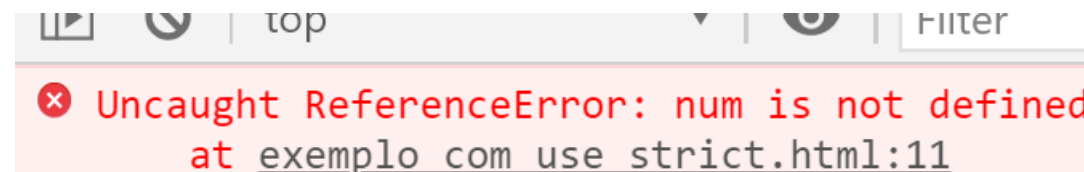
Exemplo: //faça o teste sem o "use strict"

```
num = 10;  
console.log(num); //exibe 5
```



Exemplo: //faça o teste com o "use strict"

```
"use strict";  
num = 10; //erro: num is not defined  
//console.log(num); //exibe 5
```



## Variáveis

- Toda variável deve começar com uma letra ou um underscore("\_");
- Caracteres subsequentes devem ser letras ou números;
- Não deve conter espaço em branco ou caracteres especiais;
- Não deve ser uma palavra reservada.

JavaScript é case-sensitive, logo as variáveis abaixo são todas diferentes:

- quantidade
- QUANTIDADE
- Quantidade
- QuAnTidAdE

JavaScript é uma linguagem de tipos de variáveis flexíveis, ao contrário de outras linguagens que exigem a declaração de uma variável com um tipo de dado definido.

Não precisamos definir o tipo de uma variável na declaração.

Existem 2 palavras chaves para criarmos variáveis: **var (antigo)** ou **let (moderno)**

Existe 1 palavra para criarmos as chamadas constantes: **const**

Ex:

Escopo de função e global

```
var nome="Fulano";  
ou  
var nome;
```

Escopo de bloco

```
let nome="Fulano";  
ou  
let nome;
```

Constante

```
const PI= 3,14;
```

Antes da especificação ECMAScript 2015 (ES6) JavaScript permitia apenas dois tipos de escopo: escopo global e escopo de função (global scope, function scope).

## // Escopo global

```
var vglobal = 4;

function minhafuncao() {
  // aqui podemos acessar vglobal,
  // usar ou alterar seu valor
}
```

## // Escopo de função

// aqui não podemos acessar **vfuncao**

```
function umafuncao() {
  var vfuncao = 4;
  // aqui podemos acessar vfuncao,
  // usar ou alterar seu valor
}

function outrafuncao() {
  // aqui não podemos acessar vfuncao
}
```



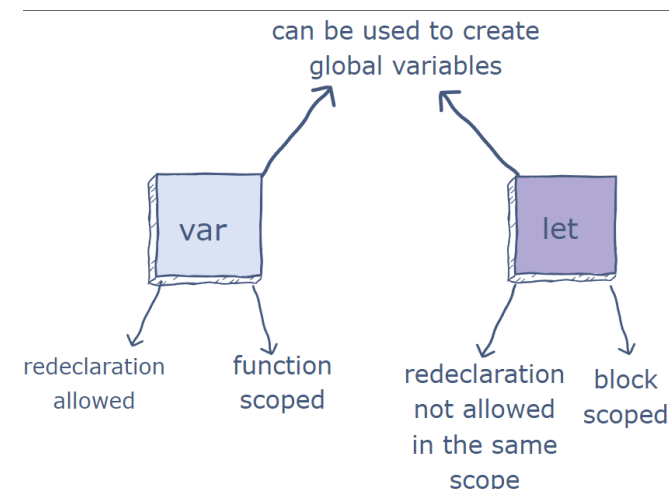
A especificação ECMAScript 2015 (ES6) JavaScript permite três tipos de escopo: escopo global, escopo de função e escopo de bloco (global scope, function scope, block scope). Foi introduzida a palavra chave **let**.

## // Escopo de bloco

```
if (...) {  
  let vbloco = 2;  
  var voutra = 4;  
}  
  
// aqui não podemos  
// acessar vbloco  
  
// mas poderíamos usar voutra  
// e isso gera confusão
```

## // Escopo de bloco

```
// aqui não podemos acessar vbloco  
  
function umafuncao() {  
  for (let vbloco=0; vbloco<=5; vbloco++) {  
    // aqui podemos acessar vbloco  
  }  
  // aqui não podemos acessar vbloco  
}  
  
function outrafuncao() {  
  // aqui não podemos acessar vbloco  
}
```



Com **const** podemos definir uma constante, uma variável cujo valor não poderá ser alterado.

## // Utilização de uma constante

```
const pi = 3.14;
pi = 3.14;    // provocará erro
pi = 2 * pi;  // também errado

// permitido porque é uma variável normal
let ang = 2 * pi;
```

## // Constante dentro de bloco

```
function estafuncao() {
  let dang = 0.0;
  for (let vbloco = 0; vbloco <= 5; vbloco++) {
    const pi = 3.14;
    console.log(pi+dang);
    dang += 0.01;
  }
  // aqui não podemos acessar pi, porque se
  // comporta com escopo de bloco
  // console.log(pi); //provocará erro
}
```

```
var x = 'valor1';  
let y = 'valor2';  
var x = 'valor3';  
let y = 'valor4'; //apresenta o erro Identifier 'y' has already been declared
```



Erro

```
let a = 1;  
a=2;  
const b=1;  
b=2; //apresenta o erro Assignment to constant variable.
```



Erro

```
console.log("Teste com VAR");  
for (var i = 0; i < 5; i++) {  
    console.log("Dentro do for"+i);  
}  
console.log("Fora do for"+i); // mostra 5
```


```
console.log("Teste com LET");  
for (let j = 0; j < 5; j++) {  
    console.log("Dentro do for"+j);  
}  
console.log("Fora do for"+j); // j is not defined
```



Erro

```
console.log("Teste das funções");
function exemploVar() {
    console.log("Ex var "+w); //undefined
    for(var w = 0; w < 5; w++) {
        console.log("Ex var "+w);
    };
    console.log("Ex var "+w); //mostra 5
};
exemploVar();
```

```
function exemploLet() {
    //console.log("Ex let "+m); //m is not defined
    for(let m = 0; m < 5; m++ ) {
        console.log("Ex let "+m);
    };
    //console.log("Ex let "+m); //m is not defined
};
exemploLet();
```



Erro

Erro

É uma boa prática inserirmos comentários em nossos códigos; em JavaScript podemos fazer isso de duas formas:

Para Comentários de várias linhas:

```
/*  
    Inserimos aqui  
    nossos comentários  
    em várias linhas  
*/
```

Para Comentários de uma única linha:

```
//Aqui inserimos somente uma linha de comentário
```

Os comentários são anotações, observações, ignoradas pelo interpretador de JavaScript.

Para você verificar a tabela abaixo, declaramos uma variável x igual a 10  
let x=10;

Operador		Exemplo	Resultado
+	Adição	<code>z = x + 10;</code>	<code>z = 20</code>
-	Subtração	<code>z = x - 10;</code>	<code>z = 0</code>
*	Multiplicação	<code>z = x * 10;</code>	<code>z = 100</code>
/	Divisão	<code>z = x / 10;</code>	<code>z = 1</code>
**	Expoente	<code>z = x**2</code>	<code>z = 100</code>
%	Módulo (resto de uma operação de divisão)	<code>z = x % 3;</code>	<code>z = 1</code>
++	Incremento antes	<code>z = ++x;</code>	<code>z = 11, x=11</code>
	Incremento depois	<code>z = x++;</code>	<code>z = 10, x=11</code>
--	Decremento antes	<code>z = --x;</code>	<code>z = 9, x=9</code>
	Decremento depois	<code>z = x--;</code>	<code>z = 10, x=9</code>

Para você verificar a tabela abaixo, declaramos duas variáveis:

```
let x=10;
```

```
let y=2;
```

Operador	Exemplo	Equivalência	Resultado
=	x = y	-	x = 2
+=	x += y	x = x + y	x = 12
-=	x -= y	x = x - y	x = 8
*=	x *= y	x = x * y	x = 20
/=	x /= y	x = x / y	x = 5
%=	x %= y	x = x % y	x = 0

Strings são sequências de caracteres, podemos representar elas de três formas em JS

- ❖ Aspas duplas "Algun texto"
- ❖ Aspas simples 'Algun texto'
- ❖ Acento grave `Algun texto`

As aspas duplas e simples não possuem diferença, contudo o acento grave sim, ele permite as chamadas templates string, veremos a seguir a ideia de uso.



Também usamos o operador "+" para concatenar Strings (textos) em JavaScript

Ex:

```
let x="Bom ";  
let y="dia";  
let z = x + y;  
// após a execução o resultado armazenará: z="Bom dia"
```

Cuidado, quando usamos o operador "+" com números e textos, o resultado é sempre texto;

```
let x=4;  
let y="a";  
let z=x+y; // o resultado será z="4a"
```

```
let x="4";  
let y="4";  
let z=x+y; // o resultado será z="44"
```

Permittem expressões embutidas, em outras palavras, podemos usar strings com varias linhas ou fazer interpolação de strings `${ }` (semelhante as concatenações).

Ex:

```
let x="Bom ";  
let y="dia";  
let z = `${x} ${y}`; // "Bom dia"
```

```
let texto = `Aqui  
    podemos  
    ter várias  
    linhas`;
```

## Tradicional

```
var a = 2;  
var b = 10;  
console.log('Resultado: ' + (a + b));
```

## Moderno

```
let a = 2;  
let b = 10;  
console.log('Resultado: ' + (a + b));  
//ou  
console.log(`Resultado: ${a + b}`);
```

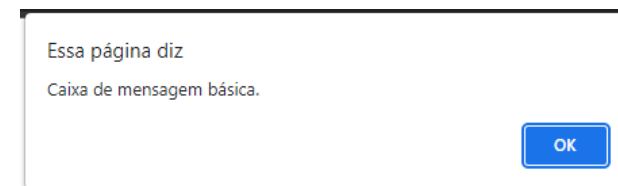
São meios básicos para entrada e saída de informação

## Caixa de alerta

Caixa de diálogo para exibir mensagens, possui somente um botão OK.

Sintaxe:

```
alert("Mensagem");
```

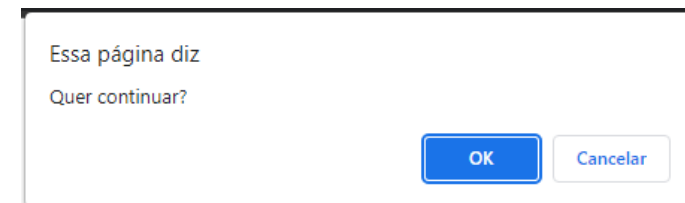


## Caixa de confirmação

Caixa de diálogo básica para tomada de decisão, possui dois botões, o botão OK (retorna true) e o botão CANCELAR (retorna false). Os valores retornados devem ser armazenados em uma variável.

Sintaxe:

```
variável = confirm("Mensagem");
```

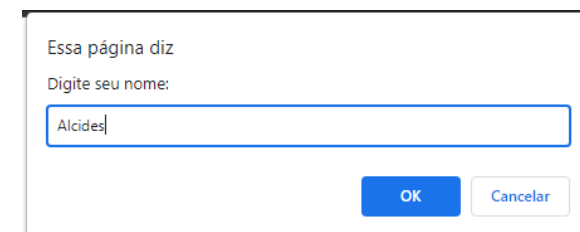


## Caixa de entrada de dados

Caixa de diálogo para entrada de dados básica, o valor deve ser armazenado em uma variável, **o valor digitado sempre será texto**, logo, para operações matemáticas precisamos converter os valores digitados para números.

Sintaxe:

```
variável = prompt("Mensagem", "valor inicial opcional");
```



Se usarmos a caixa de diálogo (prompt) ou um formulário com campos de input, os **valores lidos são sempre textos**.

Caso seja necessário realizar alguma operação matemática com os valores lidos, precisamos inicialmente convertê-los em números.

Para isso temos duas funções em JavaScript que convertem para inteiro ou para real.

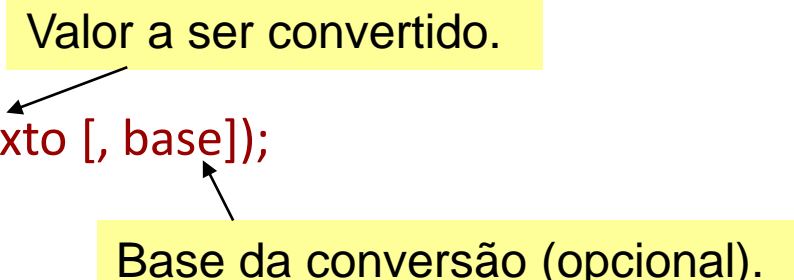
Para converter um texto em número inteiro utilizamos a função parseInt

Sintaxe:

`variável = parseInt(texto [, base]);`

Valor a ser convertido.

Base da conversão (opcional).

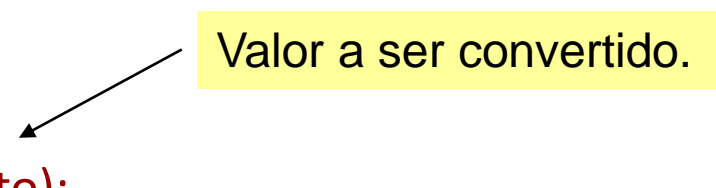
A diagram illustrating the syntax of the parseInt function. The code 'variável = parseInt(texto [, base]);' is shown in red. A yellow box labeled 'Valor a ser convertido.' has an arrow pointing to the 'texto' parameter. Another yellow box labeled 'Base da conversão (opcional).' has an arrow pointing to the optional 'base' parameter.

Para converter um texto em número real (float), utilizamos a função parseFloat.

Sintaxe:

`variável = parseFloat(texto);`

Valor a ser convertido.

A diagram illustrating the syntax of the parseFloat function. The code 'variável = parseFloat(texto);' is shown in red. A yellow box labeled 'Valor a ser convertido.' has an arrow pointing to the 'texto' parameter.

```
<script>
  "use strict";
  //modo 1
  let a;
  let aux = prompt("Digite o valor de a","");
  a = parseInt(aux);
  //modo 2
  let b = parseInt(prompt("Digite o valor de b",""));
  //mostra o cálculo
  alert(a + " + " + b + " = " + (a+b));
  //ou
  alert(`${a} + ${b} = ${a+b}`);
</script>
```

Estudamos os métodos `parseInt()` e `parseFloat()`. Esses métodos convertem respectivamente um valor `String` para inteiro ou real. No JS, esses tipos são identificados como `number`.

Outras formas de conversão:

```
let x = 23;
```

```
typeof x; //retorna o tipo number
```

```
let c = String(x);
```

```
typeof c; //retorna o tipo String
```

```
let y = Number("12.5");
```

```
let z = Number("30");
```

OBS

```
<script>
  console.log(parseInt());           // NaN
  console.log(parseInt(null));       // NaN
  console.log(parseInt(true));       // NaN
  console.log(parseInt(''));        // NaN

  console.log(Number());             // 0
  console.log(Number(null));         // 0
  console.log(Number(true));         // 1
  console.log(Number(''));          // 0
</script>
```



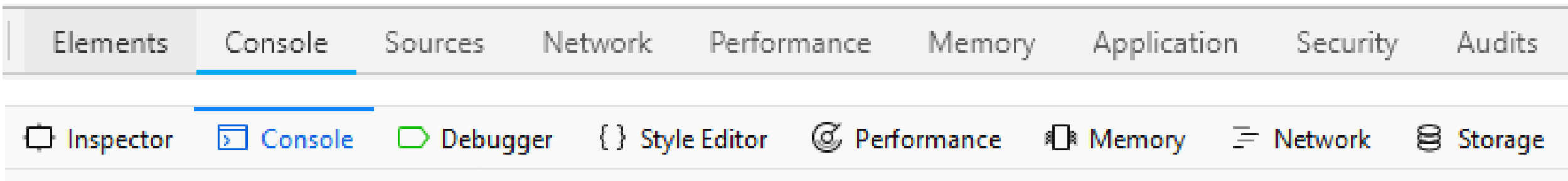
Para ativar o painel de desenvolvedor:

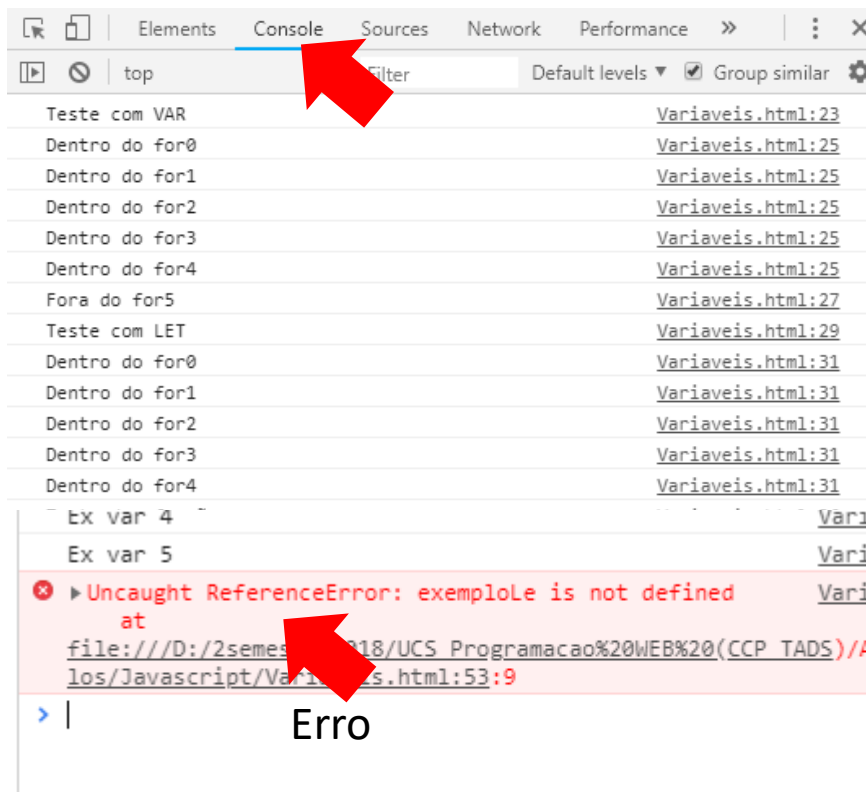
Chrome pressione a tecla F12.

Firefox F12 ou Ctrl+Shift+K

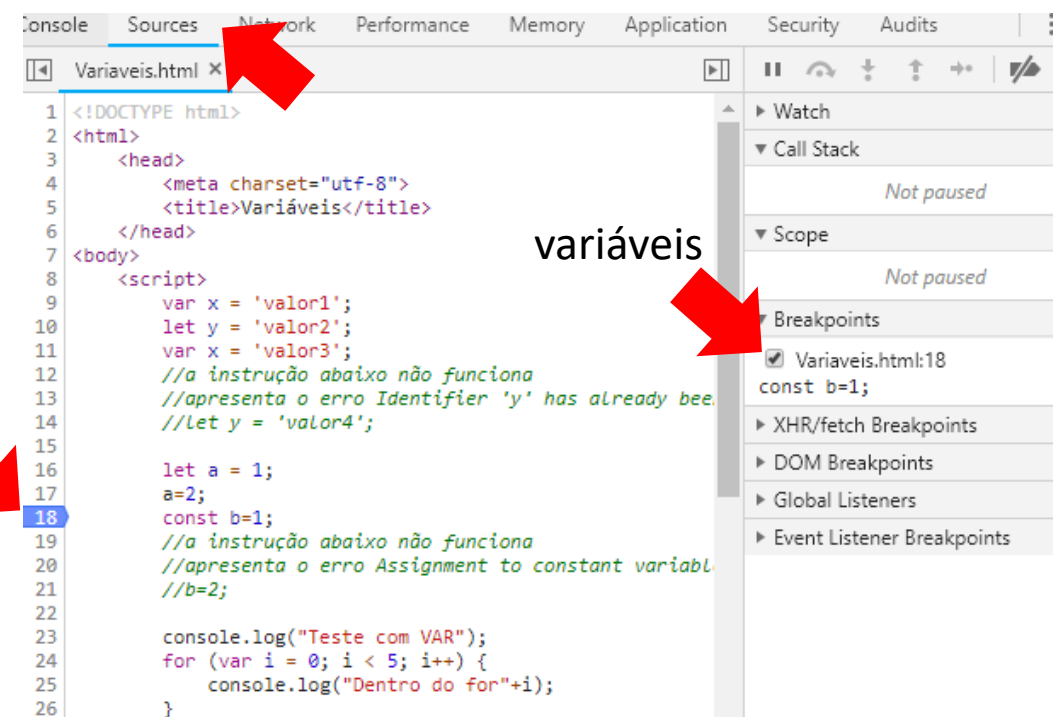
Safari: primeiro ative a ferramenta de desenvolver em Safari >> Preferências, clique em Avançado, em seguida, marque a opção “Mostrar menu Desenvolvedor na barra de menus”. Com isso será exibido o menu Desenvolvedor.

OBS: Podemos clicar com botão direito do mouse e escolher Inspecionar, isso em qualquer navegador.

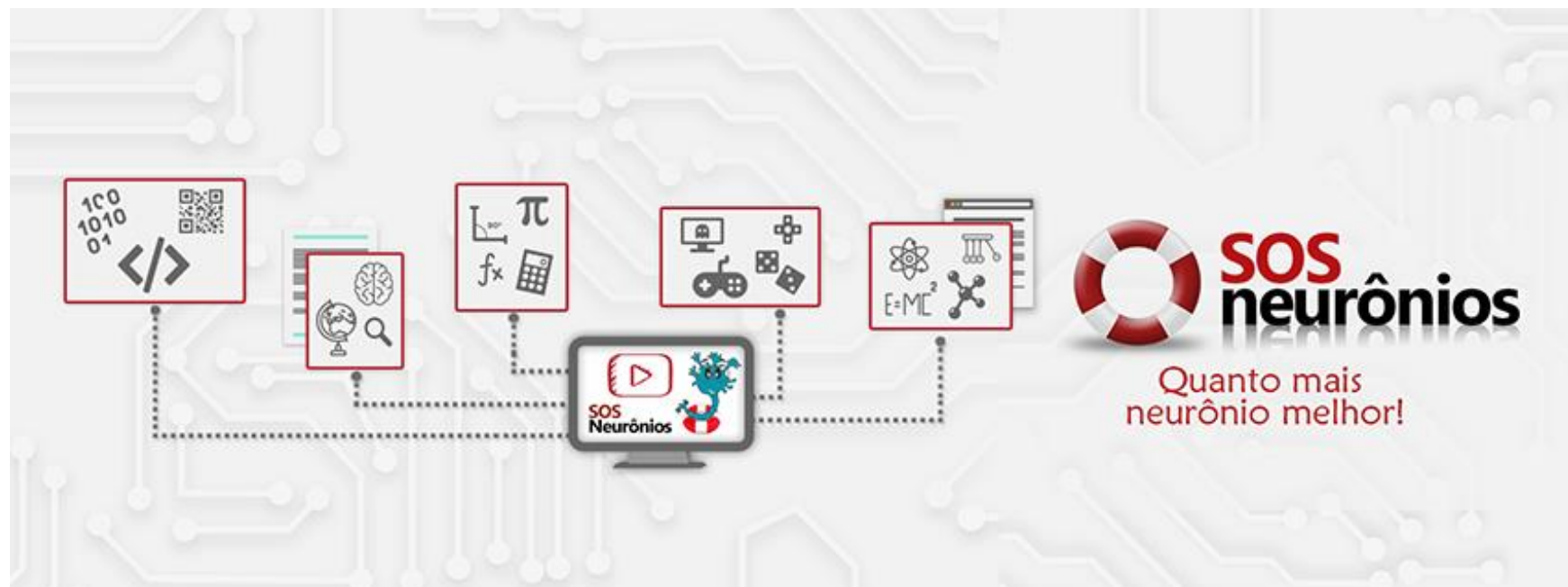




breakpoint



- 1) Uma empresa irá dar 25% de aumento para todos os funcionários, você deverá desenvolver um programa que leia o salário atual do funcionário. Após a leitura, calcule e mostre na própria página o salário atual em uma linha e em outra linha o salário com o aumento de 25%.
- 2) Crie um script que calcule e mostre na própria página a média aritmética de 4 números reais digitados pelo usuário.
- 3) Crie um script que calcule as raízes de uma equação de 2º grau  $ax^2 + bx + c$ . Supor que as raízes são reais. Para calcular a raiz você pode utilizar a função `Math.sqrt(valor)` – função própria do JavaScript. Essa função retorna o resultado da raiz quadrada que deverá ser guardado em alguma variável. Neste exemplo vamos supor que sempre teremos um delta válido.



<https://www.youtube.com/sosneuronios>

Mauricio SAMY Silva. **HTML 5 - A Linguagem de Marcação que revolucionou a WEB**. São Paulo: Novatec, 2011

MAURICIO SAMY SILVA. **Construindo Sites Com Css e (X)Html**. São Paulo: Novatec, 2007.

ERIC FREEMAN; ELISABETH FREEMAN. **Use a Cabeça! Html Com Css e Xhtml**. São Paulo: Alta Books, 2008.

DANNY GOODMAN. **Javascript e Dhtml Guia Pratico**. São Paulo: Alta Books, 2008.

MICHAEL MORRISON. **Use a Cabeça Javascript**. São Paulo: Alta Books, 2008.

Maurício Samy Silva. **JavaScript (Guia do Programador)**. São Paulo: Novatec, 2010

<http://www.w3schools.com/html/default.asp>

<http://www.w3schools.com/css/default.asp>

<http://www.w3schools.com/js/default.asp>

<https://codeburst.io/es5-vs-es6-with-example-code-9901fa0136fc>

<http://kangax.github.io/compat-table/es2016plus/>