



Developer by:
RAFAEL ANGRIZANI



— | N D I C E

Primeiros Passos na Programação	01
Tipos de Programador	02
Passos para Criar um Site	06
Ferramentas Necessárias	14
Introdução ao HTML	21
Introdução ao CSS	61
Introdução ao JAVASCRIPT	91
Introdução ao GIT & GITHUB	114
Informações ADICIONAIS	131



PLATAFORMA
DEV

PRIMEIROS PASSOS

01

LÓGICA DE PROGRAMAÇÃO

Aprenda conceitos básicos como variável, laços de repetição estruturas condicionais, independente da linguagem de programação.

02

SINTAXE DA LINGUAGEM

Após escolher a linguagem que estudará, deve-se aprender a sintaxe da linguagem, que nada mais é do que a forma como se escreve a mesma, suas palavras reservadas, etc. Após, deve-se aplicar a lógica de programação estudada à linguagem escolhida, como os laços e estruturas condicionais, dentre outras.

03

CRUD

Criar um pequeno sistema de CRUD, fazendo com que a linguagem escolhida interaja com um banco de dados, este primeiro pode ser feito de forma procedural, caso a linguagem permita.

04

POO

Aprender a forma de programar a linguagem orientada à objetos, e em seguida recrie outro CRUD reutilizando este paradigma, ao mesmo tempo aprenda GIT e GITHUB, faça download de outros projetos e os estude. Aprenda o sist. MVC.

05

FRAMEWORK

Pesquise pelo Framework de maior estabilidade do mercado relacionado à linguagem, Aprenda a desenvolver APIs. Aprenda operações básicas e avançadas, evolua e estude Inglês, facilitará em muito o seu desenvolvimento.

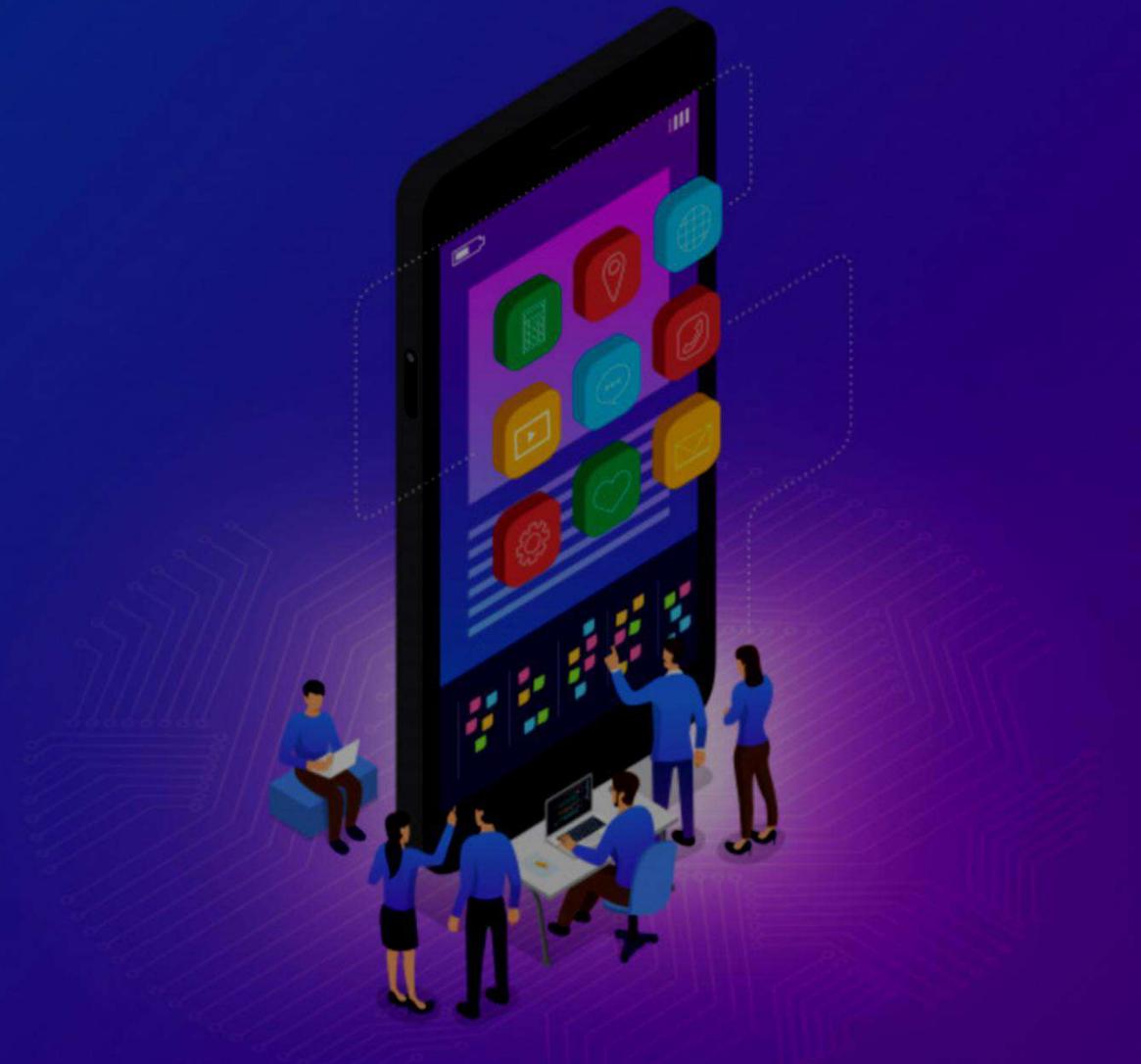
TIPOS DE PROGRAMADOR

PROGRAMADOR DESKTOP

Trabalha com sistemas locais, também chamados de sistemas off-line. Ex. Sistemas de Supermercado, farmácia, controle de estoque, etc.



designed by  freepik.com

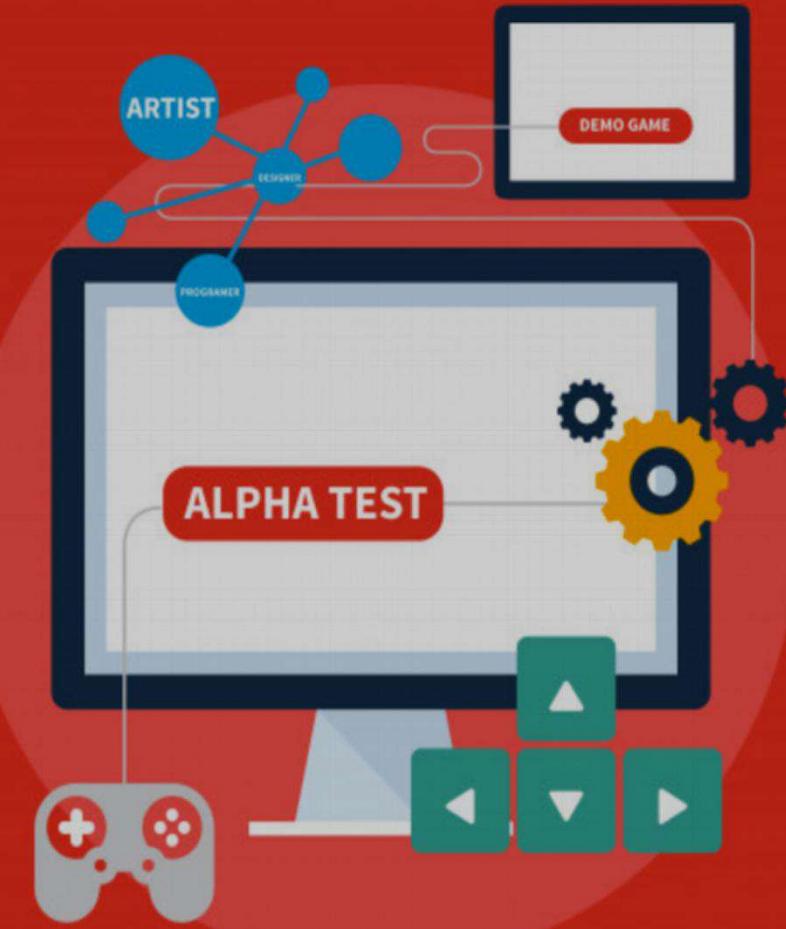


PROGRAMADOR MOBILE

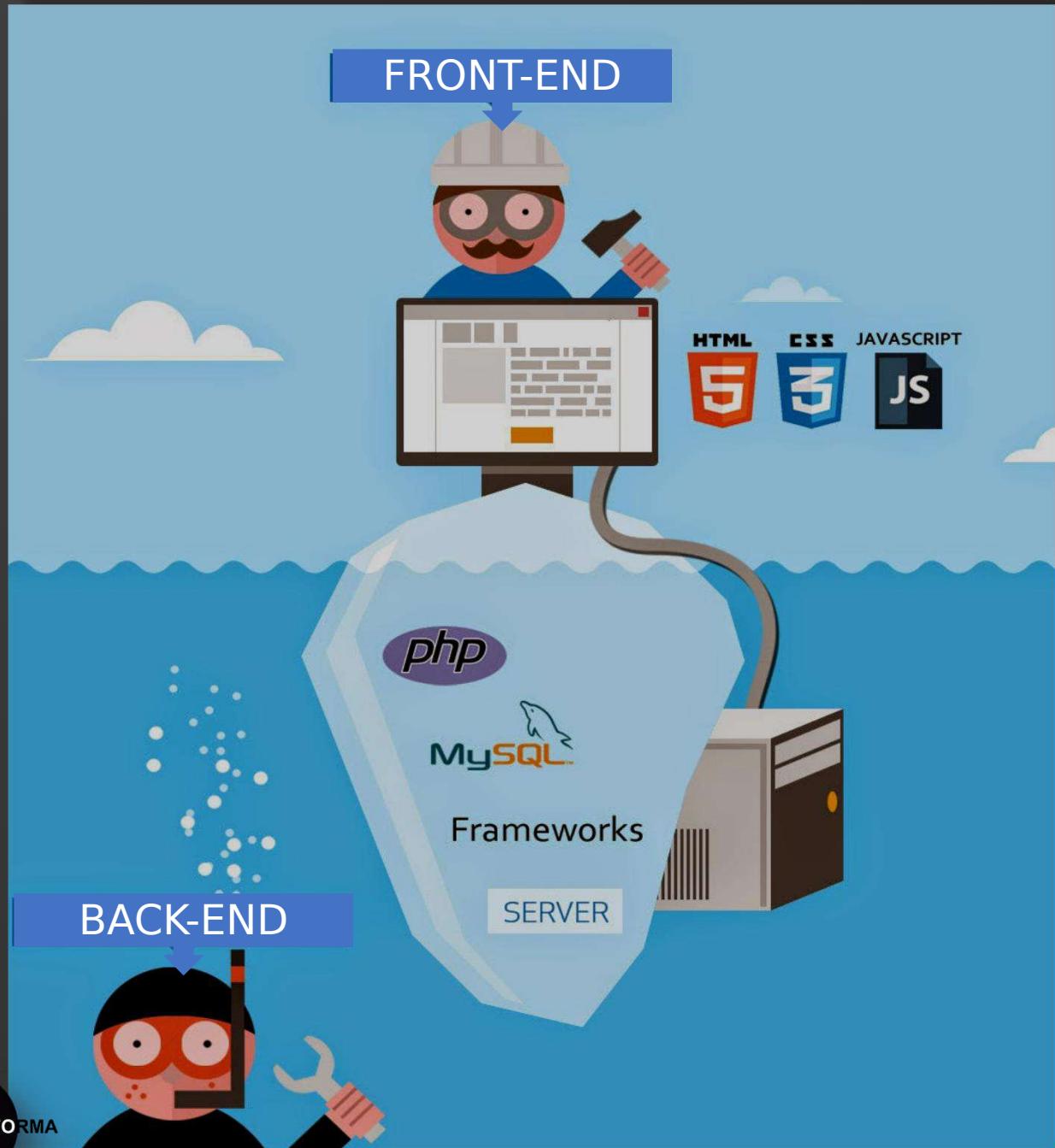
Trabalha com a criação de aplicativos para celular, desenvolve trabalhos de montagem, depuração e testes de programas, executando serviços de manutenção nos programas já desenvolvidos.

PROGRAMADOR DE GAMES

Trabalha com instruções lógicas para jogos, pode trabalhar com desenvolvimento de jogos para computador, smartphone, ou jogos que rodam em websites.



GAME DEVELOPER



PROGRAMADOR WEB

Trabalha com o desenvolvimento de sites, portais, fóruns e aplicações voltadas ao ambiente da internet. Normalmente estes serviços podem ser acessados através de um navegador e ficam hospedados em um servidor web. O programador pode ser Front-end(HTML, CSS e JS) ou Back-end(JS, PHP, PYTHON, JAVA, etc)

CRIAÇÃO DE SITES

PASSOS para



1 - PASSO

Cliente fala com a empresa para o desenvolvimento de seu site, na empresa está o analista de sistemas que coletará as informações do cliente



Web
Designer



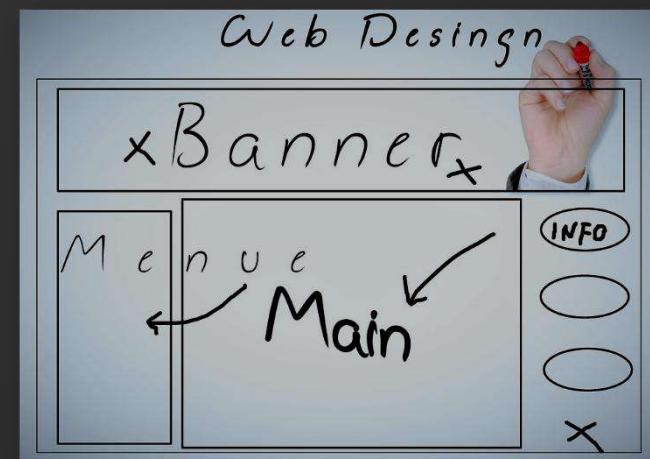
2 - PASSO

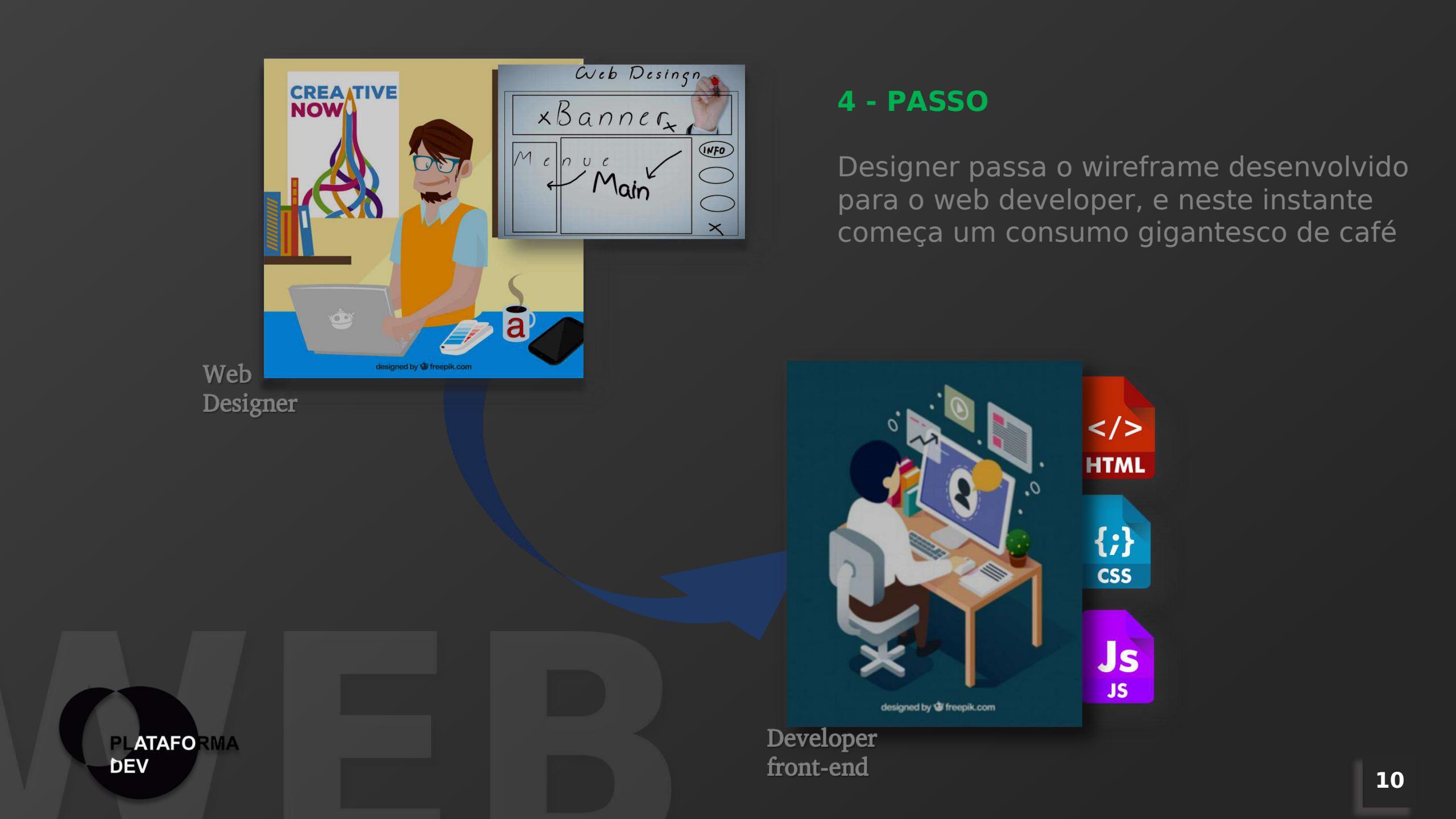
Analista repassa os dados coletados da entrevista com cliente, para o designer



3 - PASSO

Designer cria layout, logo, com base em algo já existente ou nos dados apresentados pelo analista





PLATAFORMA
DEV



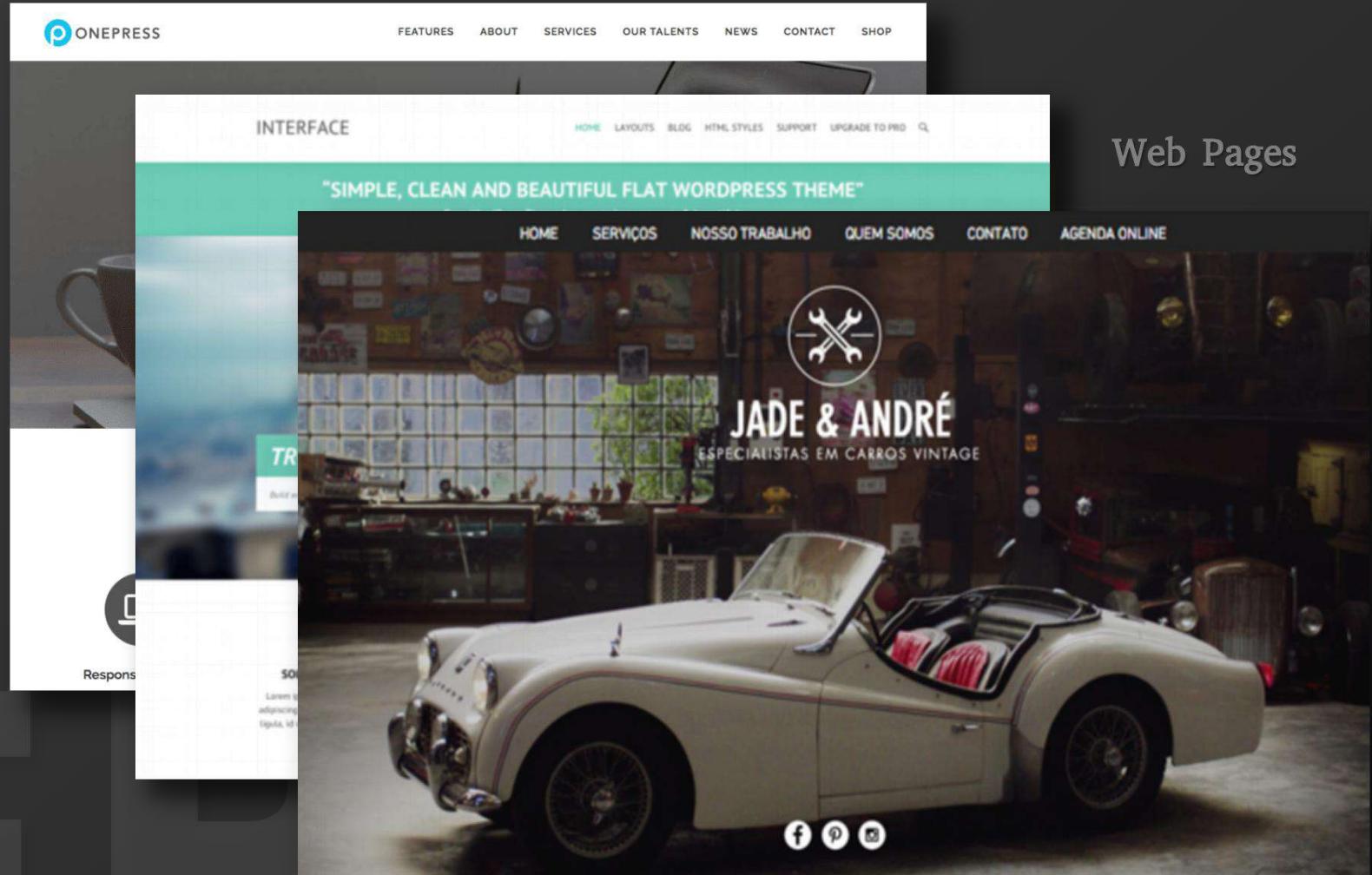
Web
Designer



Developer
front-end

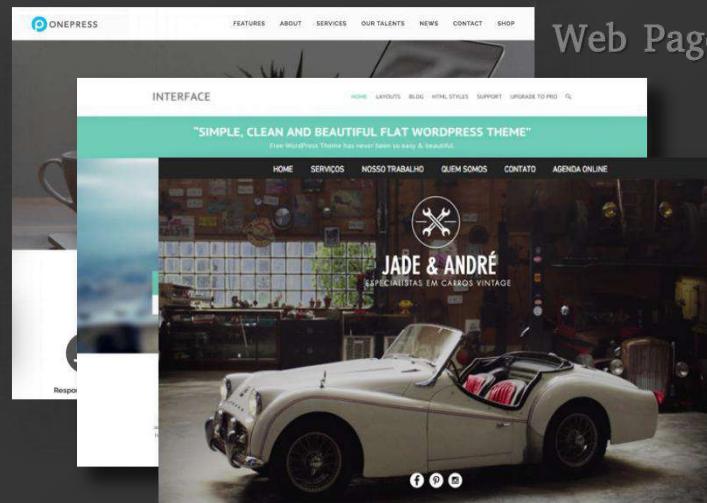
5 -PASSO

Todo café foi transformado em código, e as idéias do designer ganharam vida, então basta que envie para o desenvolvedor back-end construir a lógica.





Developer
front-end



6 - PASSO

Developer back-end realiza o trabalho necessário para a lógica da aplicação funcionar.



Developer
back-end



FIM

12





Developer by:
RAFAEL ANGRIZANI



INTRODUÇÃO e

FERRAMENTAS NESA SRIAS



FERRAMENTAS DESKTOP



Sublime Text



Visual Studio Code



Atom



IntelliJ



netbeans



Brackets



NAVEGADORES

Opera



Firefox



Google



Safari

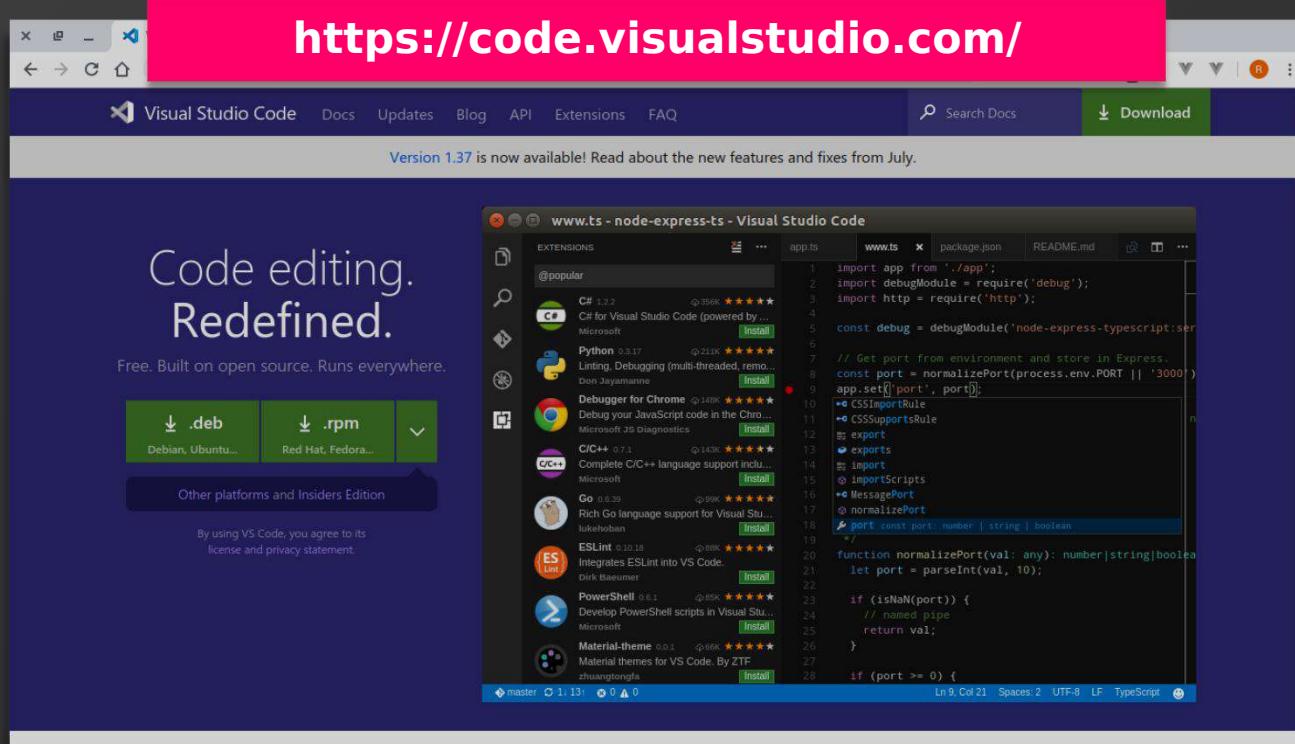


Edge



Brave

EDITOR DE TEXTO E EXTENSÕES



PLATAFORMA
DEV

Bracket Pair Colorizer [coenraads.bracket-pair-colorizer](#)
CoenraadS | ⚡ 4.124.472 | ★★★★★ | Repository | License
A customizable extension for colorizing matching brackets
[Disable](#) [Uninstall](#) *This extension is enabled globally.*

HTML Snippets [abusaidm.html-snippets](#)
Mohamed Abusaid | ⚡ 3.125.001 | ★★★★★ | License
Full HTML tags including HTML5 Snippets
[Disable](#) [Uninstall](#) *This extension is enabled globally.*

Image preview [kisstkondoros.vscode-gutter-preview](#)
Kiss Tamás | ⚡ 524.329 | ★★★★★ | Repository | License
Shows image preview in the gutter and on hover
[Disable](#) [Uninstall](#) *This extension is enabled globally.*

Live Server [ritwickdey.liveserver](#)
Ritwick Dey | ⚡ 5.352.011 | ★★★★★ | Repository | License
Launch a development local Server with live reload feature for static & dynamic pages
[Disable](#) [Uninstall](#) *This extension is enabled globally.*

Material Theme [equinusocio.vsc-material-theme](#)
Mattia Astorino | ⚡ 5.120.851 | ★★★★★ | Repository | License
The most epic theme now for Visual Studio Code
[Set Color Theme](#) [Disable](#) [Uninstall](#) *This extension is enabled globally.*

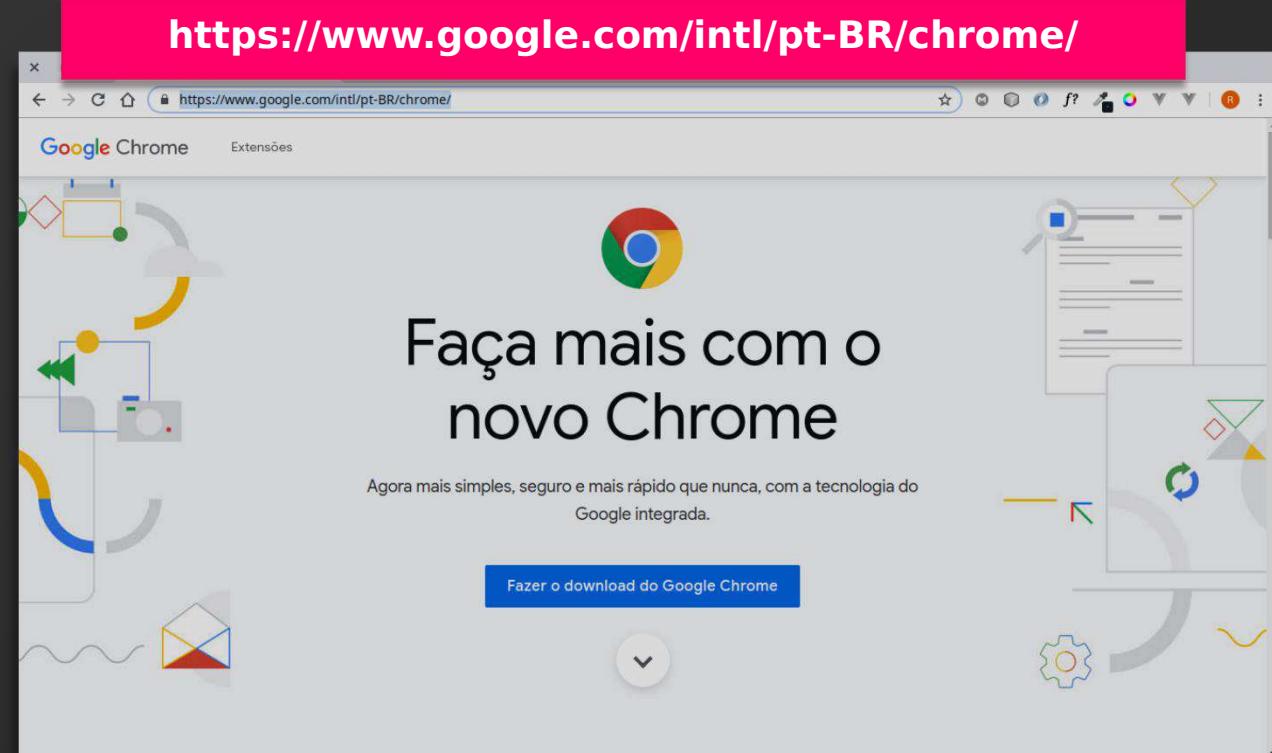
vscode-icons [vscode-icons-team.vscode-icons](#)
VSCode Icons Team | ⚡ 19.013.822 | ★★★★★ | Repository | License
Icons for Visual Studio Code
[Set File Icon Theme](#) [Disable](#) [Uninstall](#) *This extension is enabled globally.*

BROWSERS

<https://www.mozilla.org/pt-BR/firefox/new/>



<https://www.google.com/intl/pt-BR/chrome/>



LOCAL / MÁQUINA

<https://babeljs.io/>



<https://git-scm.com/>



<https://nodejs.org/en/>



<https://www.npmjs.com/>



<https://github.com/>



FONTE BIBLIOGRAFICA

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

The screenshot shows the homepage of the W3Schools HTML tutorial. The top navigation bar includes links for Home, HTML, CSS, JavaScript, SQL, PHP, Bootstrap, How To, Python, and More. A search bar and a 'Translate W3Schools' button are also present. The main content area is titled 'Tutorial em HTML5' and features a green sidebar with various HTML topics like 'HTML Introdução', 'HTML Editores', and 'HTML Atributos'. Below the sidebar, there's a section titled 'Exemplos em todos os capítulos' with a note about online editing and previewing.

<https://developer.mozilla.org/pt-BR/docs/Web/HTML>

The screenshot shows the 'HTML: Linguagem de Marcação de Hipertexto' article from MDN Web Docs. The page has a dark header with the MDN logo and a search bar. The main content area is titled 'HTML: Linguagem de Marcação de Hipertexto' and includes sections for 'Tecnologia para desenvolvedores web', 'Tópicos relacionados', 'Tutoriais', 'Referências', and a note about incomplete translation. The right side of the page contains a sidebar with related links and a note about the incomplete translation of the article.

ORGANIZAÇÃO DOS ARQUIVOS

HTML

Os arquivos html devem ser salvos com a extensão **.html** e seu arquivo principal recebe o nome de **index.html**

CSS

Os arquivos css devem ser salvos com a extensão **.css** e seu arquivo principal geralmente recebe o nome de **style.css**

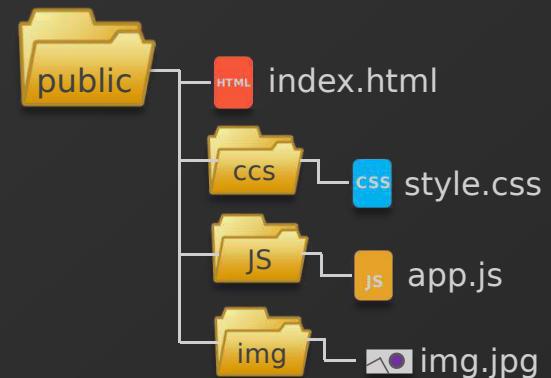
JS

Os arquivos javascript devem ser salvos com a extensão **.js** e seu arquivo principal geralmente recebe o nome de **app.js**

Estudo



Básico



— INTRODUÇÃO ao

HTML



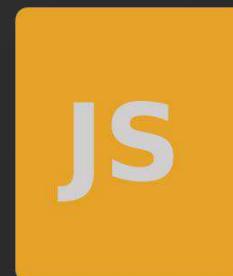
COMPOSIÇÃO BÁSICA DE UMA PÁGINA WEB



Linguagem de Marcação
Descreve a estrutura de uma página web, os seus dados e informações



Linguagem de Estilo
Personalização da forma como os dados são apresentados ao cliente



Linguagem de Programação de Scripts, gera para a página web mais dinamismo e interação com o cliente

HTML

Hyper **T**ext **M**arkup **L**anguage

1989

Linguagem de Marcação de HiperTexto

Os Elementos HTML usam TAGs para marcar o conteúdo

- └→ Tags com Corpo
- └→ Tags sem Corpo



Tim Berners-Lee

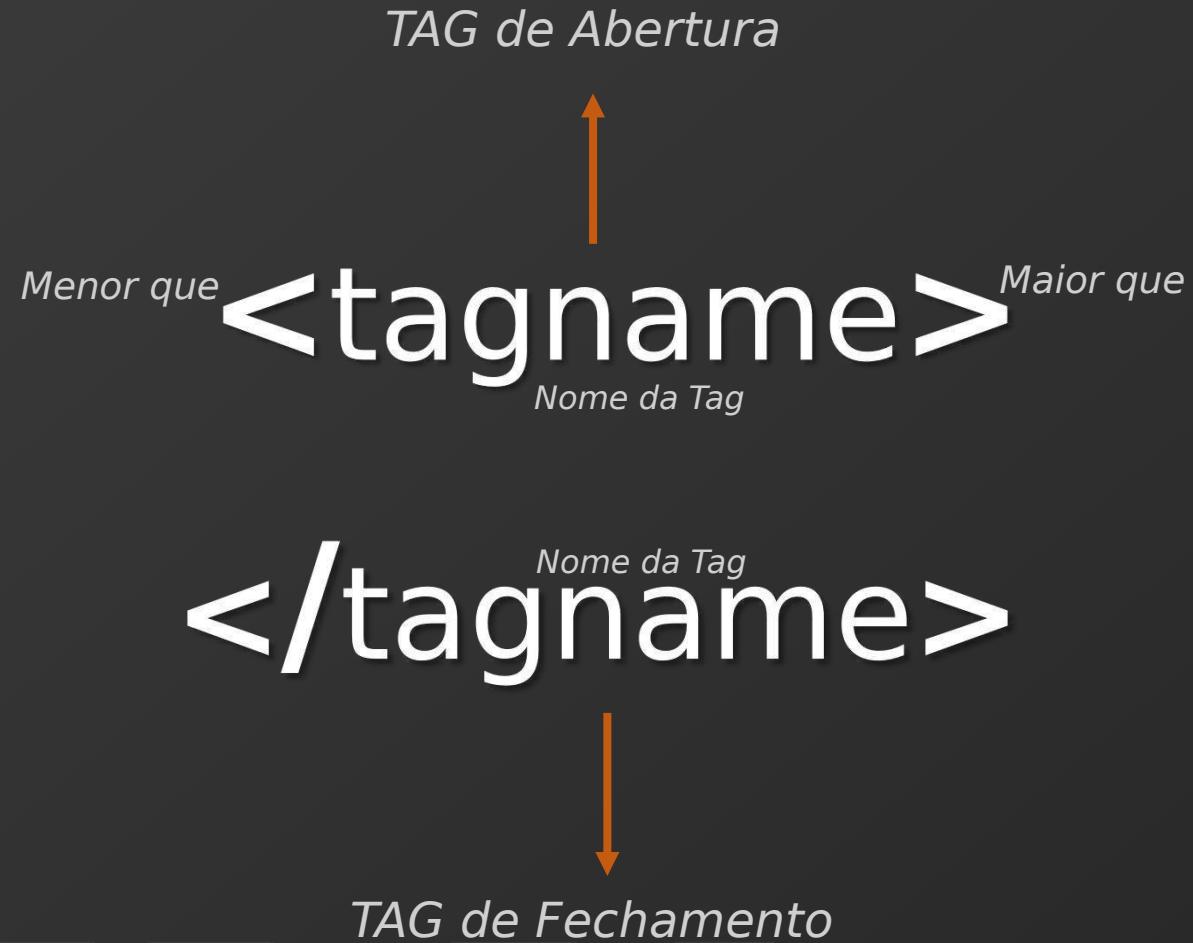
TAG SEM CORPO

Menor que *Nome da Tag* *Maior que*
<tagname>



TAG sem Corpo

TAGS COM CORPO



<tagname> → Abertura da TAG



Tags dentro de Tags, (aninhamento de tags)
Tag Pai e dentro Tags Filhas

</tagname> → Fechamento da TAG (/)

<tagname> → *Abertura da TAG*

Conteúdo
Conteúdo
Conteúdo
Conteúdo
Conteúdo
Conteúdo

*Conteúdo a ser mostrado
Podendo ser imagens, vídeo, texto, etc*

</tagname> → *Fechamento da TAG*

FECHAMENTO DAS TAGS



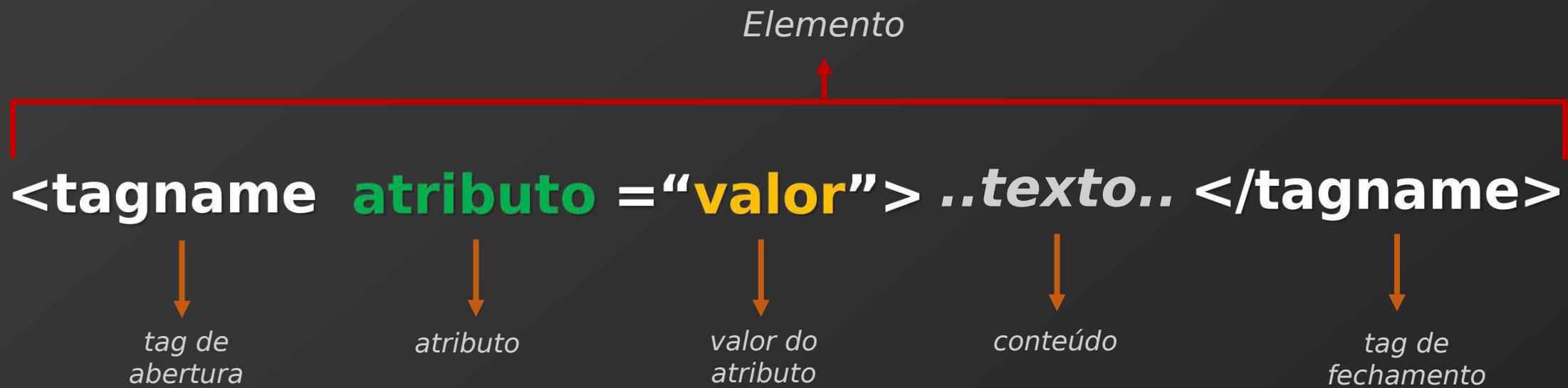
```
<tag1><tag2><tag3>..texto..</tag1></tag2></tag3>
```



```
<tag1><tag2><tag3>..texto..</tag3></tag2></tag1>
```

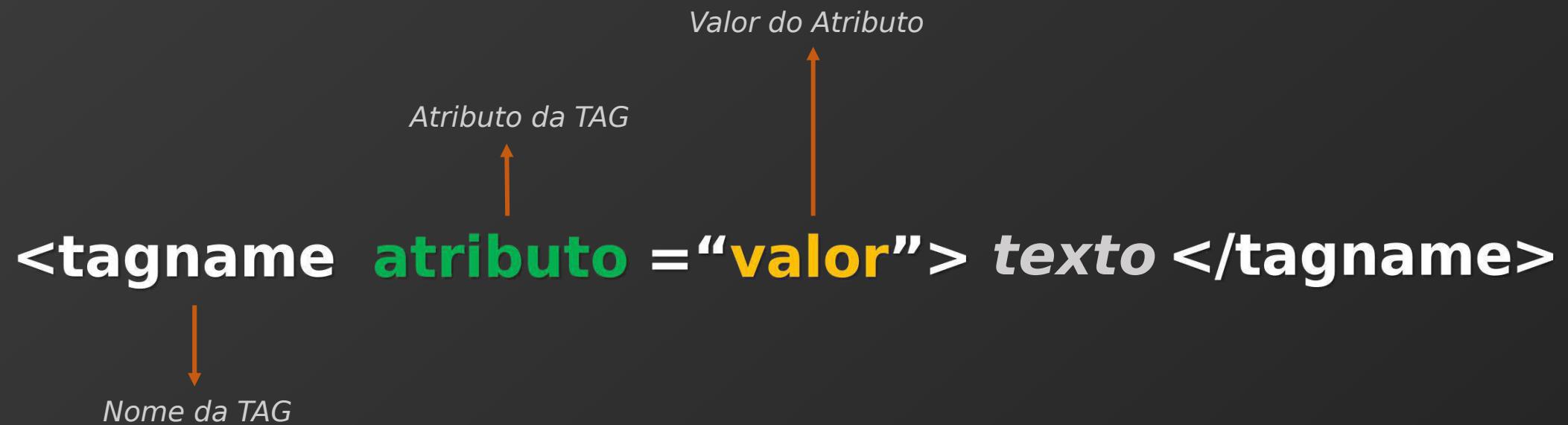
ELEMENTO HTML

Um elemento refere-se a um par de tags e seu conteúdo, ou a uma tag "vazia", que não requer tag de fechamento ou conteúdo, basado na idéia de aninhamentos, alguns elementos podem ter ainda outros elementos

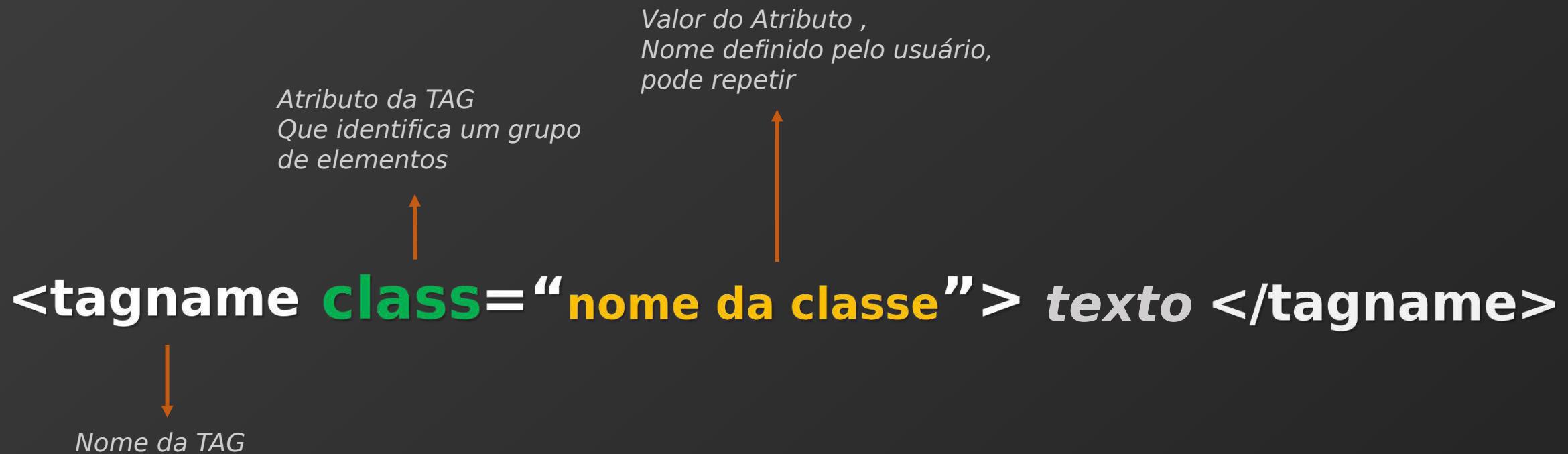


ATRIBUTOS

Atributos são informações adicionais sobre o elemento, e devem ser colocados sempre na tag de abertura



ATRIBUTOS IDENTIFICADORES



```
<tagname id=“nome do id”> texto </tagname>
```

Nome da TAG

*Atributo da TAG,
Identificador ÚNICO
Do elemento*

*Valor do Atributo ,
Nome definido pelo usuário,
Não deve repetir, deve
manter-se único*

ESTRUTURA BÁSICA HTML

Interpretação do código

Define o cabeçalho do site, meta informações sobre a página, links para arquivos externos, SEO)

Define o corpo do site, conteúdo (informações visíveis)

```
<!DOCTYPE html> → Informa ao navegador que será um documento do tipo HTML5  
<html lang="pt-br"> → Tag que inicia o nosso Documento HTML, com atributo lang que define a linguagem em que o site será desenvolvido  
  
<head>  
  <meta charset="UTF-8"> → aceitará meta caracteres, como acentos, cedilhas, etc  
  <title> Titulo do Site </title> → Título do site  
</head>  
  
<body>  
  <!-- Aqui criamos o site -->  
</body>  
  
</html> → Tag que encerra o nosso documento HTML
```



ALGUMAS DAS TAGS MAIS USADAS

<h1> ..Título.. </h1>

*Sendo que <h1>, possui maior relevância para o motor de busca dos navegadores, <h2> menos relevância do que h1, e assim sucessivamente, lembrando que vai do **<h1> até <h6>***

<p> ..text.. </p>

Define um parágrafo em um trecho de texto

**
**

Define uma quebra de linha no documento

<!-- comentário -->

Define um comentário no documento, e não é interpretado pelo navegador

```
<!-- início do post -->
<h1> Título do Post </h1>
<h2> Subtitulo do Post </h1>
<p>
    Agora aqui temos um
    parágrafo, do nosso post
</p>
<!-- final do post -->
```

Titulo do Post

Subtitulo do Post

Agora aqui temos um
parágrafo, do nosso post

Titulo do Post

Agora aqui temos um
parágrafo, do nosso post

Agora aqui temos um outro
parágrafo, do nosso post

```
<h1> Título do Post </h1>
<!-- final titulo, e abaixo o texto -->
<p>
    Agora aqui temos um
    parágrafo, do nosso post
</p>
<p>
    Agora aqui temos um outro
    parágrafo, do nosso post
</p>
```

`<a> ..link.. `

Define um parágrafo em um trecho de texto

href = “endereço”

Atributo usado para especificar o endereço do link,
Este endereço pode ser **Relativo** links na mesma página, ou
Absoluto com endereços completos geralmente externos

target = “_blank”

Atributo usado para abrir uma nova janela,
Este atributo determina que seja aberta uma nova aba no
navegador, para ser apresentado o conteúdo relacionado ao
link

Define uma imagem

src = “*imagem.jpg*”

Atributo que especifica o arquivo de origem, caso necessário, indicar a pasta de origem também se estiver em outro diretório

width = “*tamanho*”

Atributo que define um tamanho de largura para imagem

height = “*tamanho*”

Atributo que define um tamanho de altura para imagem

alt = “*texto*”

Atributo define um texto alternativo, caso a imagem não seja carregada

```
<!-- início do card -->
```

```
<h2> Titulo do Card </h1>
```

```

```

```
<p> Legenda da Imagem </p>
```

```
<a href="https://google.com/img"> Ver Mais </a>
```

```
<!-- final do card -->
```

Titulo do Card

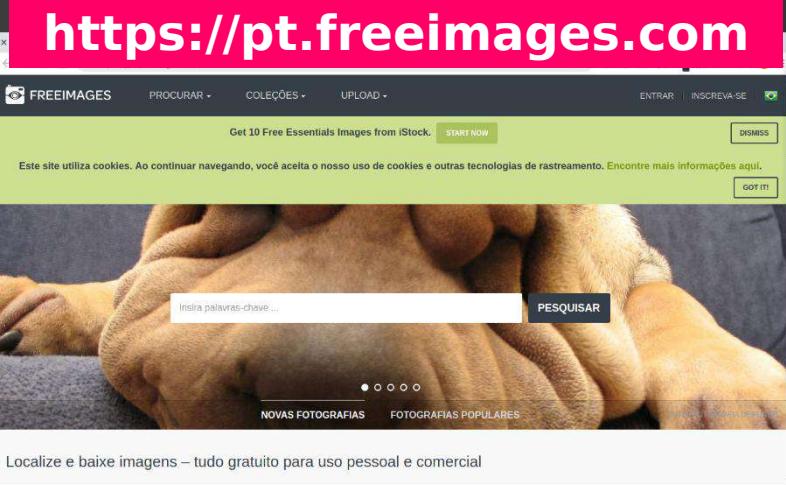


Legenda da Imagem

Ver Mais

OBS: Por padrão o navegador não carregará os elementos desta maneira, precisamos do CSS

BANCO DE IMAGENS GRÁTIS



`<hr>`

É usado para separar o conteúdo, ou definir uma alteração no fluxo do mesmo

`<pre> ..texto.. </pre>`

Texto pré-formatado, conservando espaços e quebras de linha

` ..texto.. `

Texto em negrito

`<i> </i>`

Icones

** ..texto.. **

Texto em negrito com significado especial

<mark> ..texto.. </mark>

Texto marcado

<small> ..texto.. </small>

Texto pequeno

** ..texto.. **

Texto excluído

_{..texto..}

Texto subscrito

^{..texto..}

Texto sobrescrito

** ..texto.. **

Texto enfatizado

<q> ..texto.. </q>

Citação curta, geralmente os navegadores colocam áspas

<blockquote> ..texto.. </blockquote>

Define uma seção de texto que é citada de outra fonte, os navegadores aplicam um recuo ao texto

<abbr> ..texto.. </abbr>

Define uma abreviação ou um acrônimo

<address> ..texto.. </address>

Informações de contato de um autor de um documento

<cite> ..texto.. </cite>

Define o título de um trabalho

<table> ..itens.. </table>

Define uma tabela no documento

<caption> ..texto.. </caption>

Legenda de uma tabela

<thead> ..itens.. </thead>

Agrupa o conteúdo do cabeçalho, geralmente usado com **<tbody>** e **<tfoot>**, e depois do **<caption>**

<tbody> ..itens.. </tbody>

Agrupa o conteúdo do corpo de uma tabela, geralmente usado com **<tbody>** e **<tfoot>**, e depois do **<caption>**

<tfoot> ..itens.. </tfoot>

Agrupa o conteúdo do rodapé, geralmente usado com **<tbody>** e **<tfoot>**, e depois do **<caption>**

<th> ..texto.. </th>

Define uma célula de cabeçalho

<tr> ..itens.. </tr>

Define uma linha na tabela

<td> ..dados.. </td>

Define uma célula padrão de dados na tabela

```
<table>
  <thead>
    <tr>
      <th> Header 1 </th>
      <th> Header 2 </th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td> Footer 1 </td>
      <td> Footer 2 </td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td> Body 1 </td>
      <td> Body 2 </td>
    </tr>
  </tbody>
</table>
```

Header 1	Header 2
Body 1	Body 2
Footer 1	Footer 2

OBS: Por padrão o navegador não carregará os elementos com esta cores, precisamos do CSS



** ..itens.. **

Define uma lista desordenada

** ..itens.. **

Define uma lista ordenada

<dir> ..itens.. </dir>

Define uma lista de diretórios

** ..dados.. **

Define os itens de uma lista (ul, ol, dir)

```
<ol>
```

```
    <li> item 01 </li>
    <li> item 02 </li>
    <li> item 03 </li>
    <li> item 04 </li>
```

```
</ol>
```

```
1. item 01
2. item 02
3. item 03
4. item 04
```

```
<ul>
```

```
    <li> item 01 </li>
    <li> item 02 </li>
    <li> item 03 </li>
    <li> item 04 </li>
```

```
</ul>
```

```
. item 01
. item 02
. item 03
. item 04
```

<form> </form>

↓

Uma Seção de um documento que permite ao usuário através de controles interativos e campos enviar informações à um servidor

action = “nome_arquivo”

↓

Nome do arquivo que receberá os dados do formulário e os processará, quando este for enviado

method = “nome_método”

Nome do método HTTP que será usado para envio do formulário, pode ser GET (via URL), ou POST(incorporado no corpo do formulário)

<fieldset> </fieldset>

Agrupa elementos dentro de um formulário

<label> </label>

Represents a legend for a specific type of input data in a form.

for = “id_campo”

Defines which input data this label will associate with, specifically the `id` of the input element.

<textarea></textarea>

Represents a large text input field, such as a comment or return form.

<input>

↓
Represents a legend for a specific type of input field in a form.

type = "tipo de campo"

↓
Defines the type of field, the default is 'text', but we can change it to 'radio', 'checkbox', 'color', 'date', 'email', 'file', 'submit', 'image', 'number', 'password', etc.

maxlength = "numero"

↓
Defines the maximum number of characters

minlength = "numero"

↓
Defines the minimum number of characters

placeholder = "texto"

↓
Defines a hint for the user, what will be expected in the field

required = "required"

↓
Defines that filling the field is mandatory, otherwise the form is not sent

```
<form action="processa.php" method="post">
  <fieldset>

    <label for="id_nome"> Nome: </label>
    <input type="text" placeholder="Seu Nome" id="id_nome" required="required" >

    <label for="id_email"> E-mail: </label>
    <input type="email" placeholder="Seu E-mail" id="id_email" required="required" >

    <textarea> Mensagem </textarea>

    <input type="submit">

  </fieldset>
</form>
```

The image shows a simple HTML form presented as a modal or a card. It contains three input fields: a text input for 'Nome' with placeholder 'Seu Nome', an email input for 'E-mail' with placeholder 'Seu E-mail', and a textarea for 'Mensagem'. Below these fields is a large, prominent 'Enviar' button.

OBS: Por padrão o navegador caregará os elementos do formulário um ao lado do outro

BLOCK / INLINE

- Elementos de bloco ocupam 100% da largura da linha em que foram criados

- Elementos Inline, ocupam apenas o espaço do conteúdo correspondente

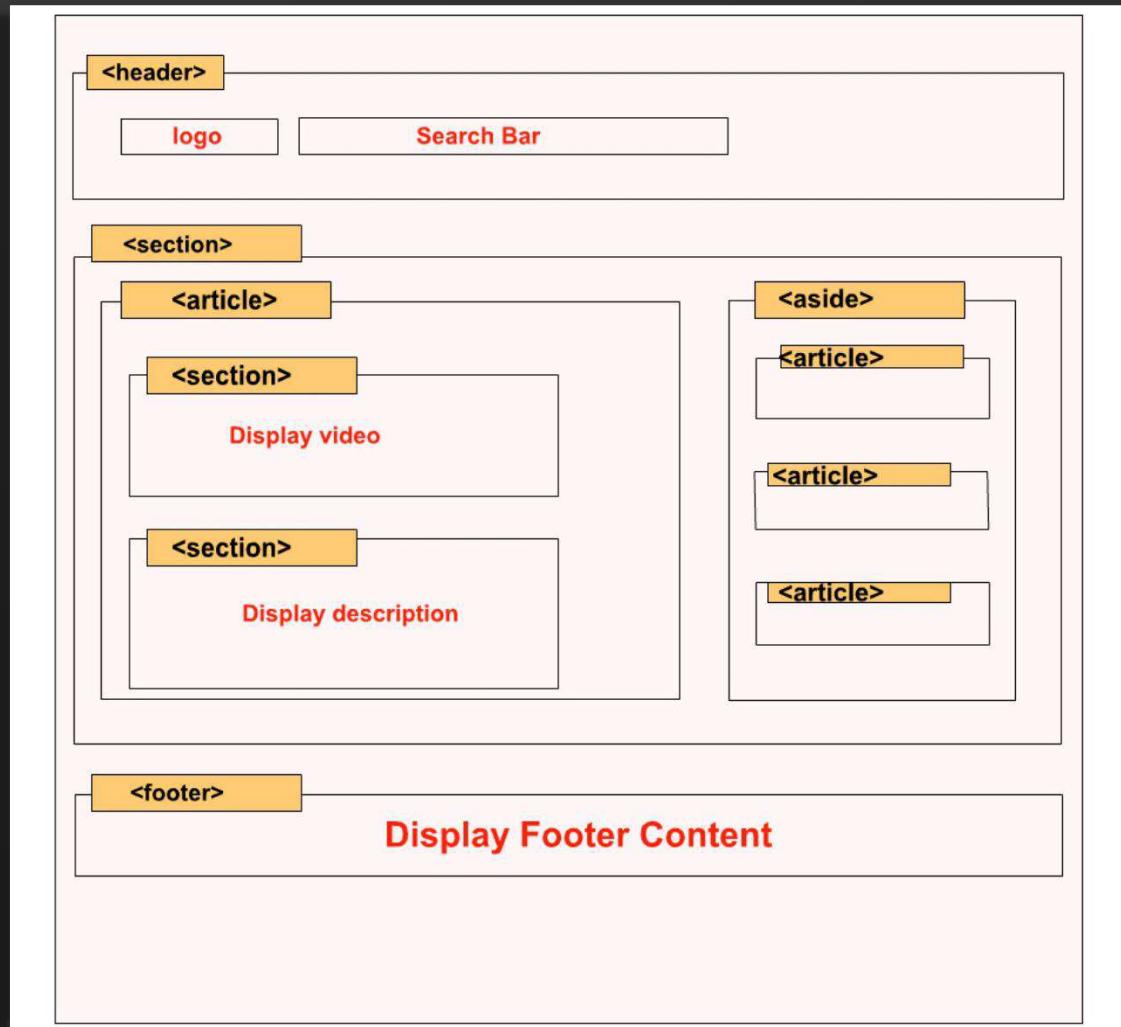
The screenshot shows a web browser window with the URL <https://www.plataformadev.com.br>. The page content is as follows:

- A large green rectangular box labeled "Elemento de Bloco 01".
- A large green rectangular box labeled "Elemento de Bloco 02".
- A horizontal row of three yellow rectangular boxes labeled "Inline 01", "Inline 02", and "Inline 03".
- A large green rectangular box labeled "Elemento de Bloco 03".
- A yellow rectangular box labeled "Inline 04" followed by the text "display:block" in a smaller font.
- A green rectangular box labeled "Elemento de Bloco 04" with the text "Display:inline, + propriedades" below it.
- A green rectangular box labeled "Elemento de Bloco 05" with the text "Display:inline , + propriedades" below it.

Através de prorpiedades CSS podemos inverter

HTML SEMÂNTICO

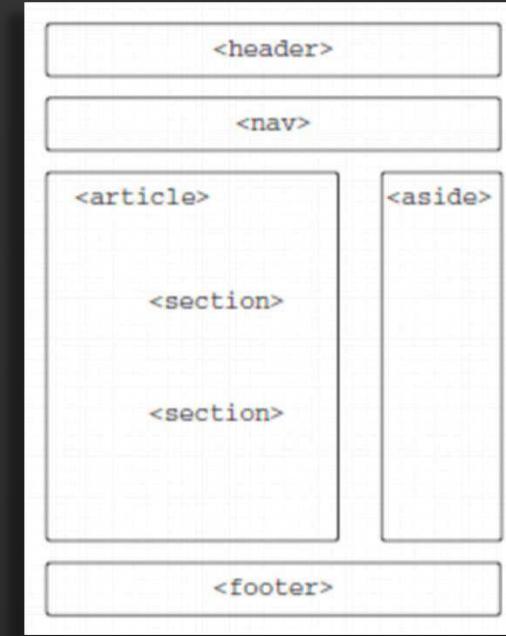
Descreve o significado do conteúdo, através de tags semânticas, tornando-o mais claro para o browser e outras engines que processam esta informação (semântico = significado)



<HEADER>

Representa o cabeçalho de um documento pode conter mais de 01 por documento, não pode ser filho da Tag Footer

```
<article>
  <header>
    <h1> Mais importante Título aqui </h1>
    <h3> Menos importante titulo aqui </h3>
    <p> Alguma informação adicional aqui </p>
  </header>
  <p> Informações pertinentes aqui </p>
</article>
```



O footer não deve conter cabeçalhos, já que não atendem às necessidades semânticas do footer esta finalidade

```
<header>
  <h1> Mais importante Título aqui </h1>
  <h3> Menos importante titulo aqui </h3>
  <p> Alguma informação adicional aqui </p>
</header>
```

<NAV>

Representa o agrupamento Principal de links de navegação

```
<nav>
  <ul>
    <li><a href="#"> pagina 1 </a></li>
    <li><a href="#"> pagina 2 </a></li>
    <li><a href="#"> pagina 3 </a></li>
    <li><a href="#"> pagina 4 </a></li>
  </ul>
</nav>
```



No exemplo acima criamos a tag NAV e colocamos dentro dela links para outras páginas organizados dentro de uma lista desordenada, pois semanticamente não possui uma organização específica e nem uma maior importância entre um e outro, por isto a lista desordenada, entretanto poderíamos ter colocado apenas os links sem qualquer problema, isso depende do desenvolvedor

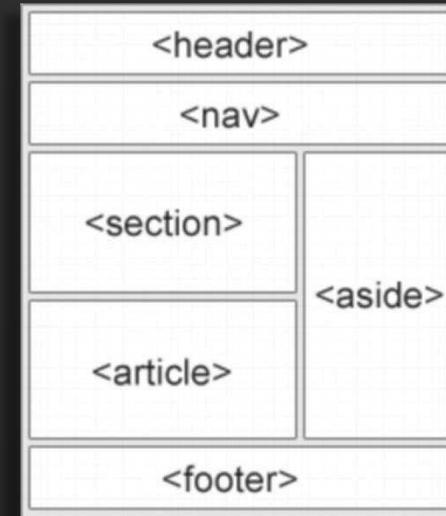


<SECTION>

Representa uma seção de conteúdos em comum dentro do documento, como capítulos, cabeçalhos, rodapés e geralmente possui título entre h1 até h6

```
<section id="noticias">  
  
  <section id="esporte">  
    ...  
  </section>  
  <section id="politica">  
    ...  
  </section>  
  <section id="educacao">  
    ...  
  </section>  
  
</section>
```

```
<section>  
  <h1> Mais importante Título aqui </h1>  
  <h3> Menos importante titulo aqui </h3>  
  <p> Alguma informação adicional aqui </p>  
</section>
```

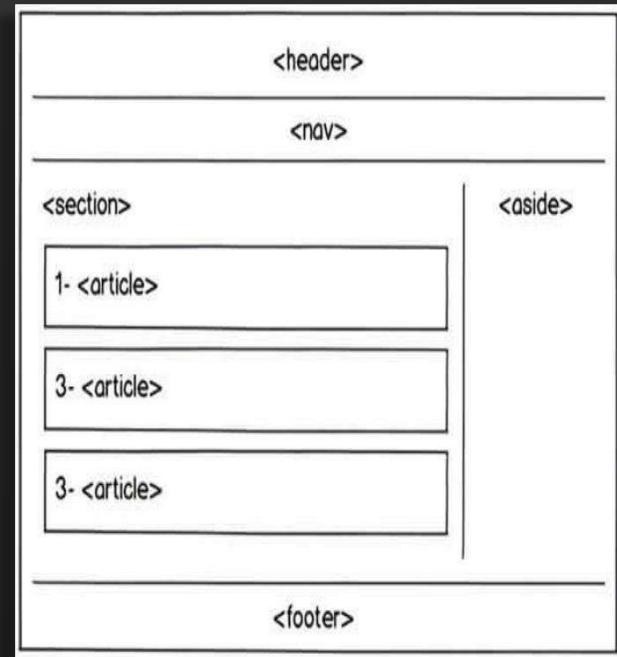


<ARTICLE>

Declara um conteúdo que não precisa de outro para fazer sentido, pois especifica um conteúdo independente, deve fazer sentido por si só

```
<article>
  <h1> Mais importante Título aqui </h1>
  <p> Alguma informação adicional aqui </p>
</article>
```

```
<article>
  <h1> Mais importante Título aqui </h1>
  <p> Alguma informação adicional aqui </p>
</article>
```

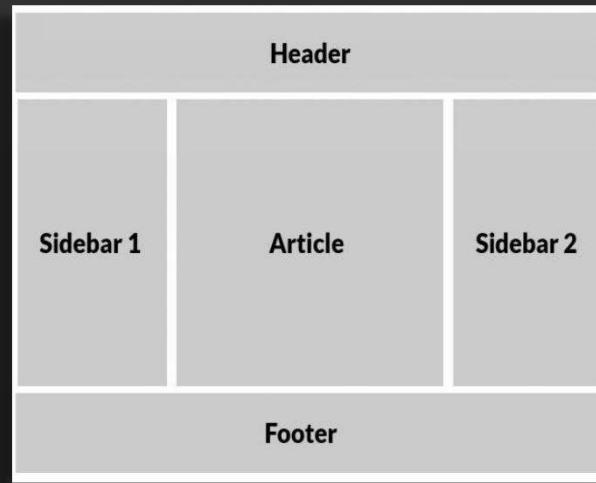


```
<article>
  <h1> Mais importante Título aqui </h1>
  <p> Alguma informação adicional aqui </p>
</article>
```

<ASIDE>

Representa um conteúdo de apoio, ao conteúdo principal, pode ser colocado como uma barra lateral em um artigo, geralmente coloca-se links para outros conteúdos complementares

```
<aside>
  <ul>
    <li><a href="#"> link 1 </a></li>
    <li><a href="#"> link 2 </a></li>
    <li><a href="#"> link 3 </a></li>
    <li><a href="#"> link 4 </a></li>
  </ul>
</aside>
```



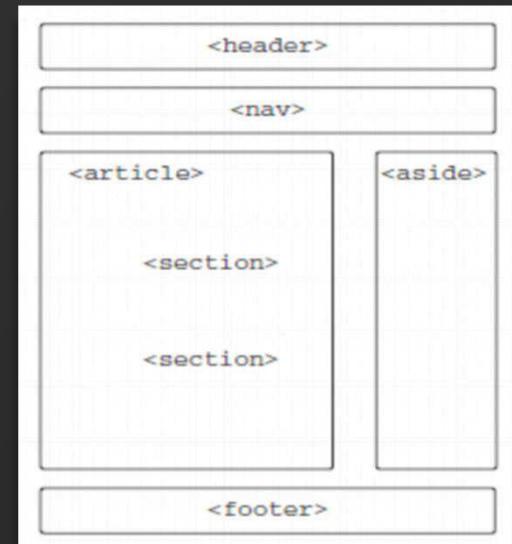
```
<p> Alguma informação aqui </p>

<aside>
  <h1> Título sobre a informação aqui </h1>
  <p> Alguma descrição sobre a informação aqui </p>
</aside>
```

<FOOTER>

Representa o rodapé de um documento ou seção, nele pode conter elementos referentes ao desenvolvedor, mapa do site, formulários de contato, etc. ou informações sobre sua seção contida.

```
<footer>
<h1> Mapa do Site </h1>
  <ul>
    <li><a href="#"> pagina 1 </a></li>
    <li><a href="#"> pagina 2 </a></li>
    <li><a href="#"> pagina 3 </a></li>
    <li><a href="#"> pagina 4 </a></li>
  </ul>
</footer>
```



— INTRODUÇÃO ao

css



CSS

Cascade **S**tyle **S**heet

Folha de Estilo em Cascata

Usa SELETORES para estilizar o conteúdo da página web



Håkon Wium Lie



1996

LOCALIZAÇÃO CÓDIGO CSS

Código CSS sendo chamado de arquivo externo

HTML

```
<link rel="stylesheet" href=".style.css" >
```

css

```
h1{  
    color: blue;  
}
```

Código CSS dentro do documento HTML

```
<style>  
  
    h1{  
        color: blue;  
    }  
  
</style>
```

Código CSS dentro da tag HTML

```
<h1 style="color:blue"> Titulo </h1>
```

SELETOR DE TAG

tag HTML

<tagname> conteúdo </tagname>

seletor CSS

```
tagname {  
    Color: red;  
}
```

Propriedade

Valor da Propriedade

resultado

conteúdo

SELETOR DE ID

tag HTML

<tagname id = “nome do id” > conteúdo </tagname>

seletor CSS

#nome do id{

Color: red;

}

Propriedade

Valor da Propriedade

resultado

conteúdo

SELETOR DE ATRIBUTO EXATO

tag HTML

```
<input type="text" >
```

seletor CSS

```
input [type="text"] {  
    color: red;  
}
```

Propriedade

Valor da Propriedade

resultado

conteúdo

SELETOR HIERÁRQUICO

tag HTML

```
<h1> Título </h1>
    <p> Isso é um parágrafo
```

seletor CSS

```
H1 P {
    color: red;
}
```

Propriedade

Valor da Propriedade

resultado

Título

Isso é um parágrafo

ESTUDO DAS CORES

Ao se trabalhar com cores no CSS, temos algumas alternativas de declaração, podemos usar a notação:

HEXADECIMAL - usa-se no formato # rr gg bb (red, green, blue) são valores hexadecimais entre 00 e ff (igual decimal 0 até 255)

RGB - (red, green, blue), cada parâmetro define a intensidade da cor entre 0 e 255, pode usar o valor de alfa também com RGBA(red, green, blue, alfa) o alfa vai de 0,0(totalmente transparente) até 1,0(nenhum pojco transparente)

HSL - (matiz, saturação e luminosidade) matiz é um grau na roda de cores que vai de 0 (vermelho) até 360(azul), a saturação é um valor percentual entre 0% (cinza) até 100% (preto cor mais pura), e a luminosidade é uma porcentagem entre 0% (preto) e 100% (branco), pode usar o valor de alfa também com HSLA (matiz, saturação, luminosidade e alfa) o alfa vai de 0,0 (totalmente transparente) até 1,0 (nenhum pojco transparente)

NAME - Você pode simplesmente escrever o nome da cor que deseja em linguagem inglesa

EXEMPLO:

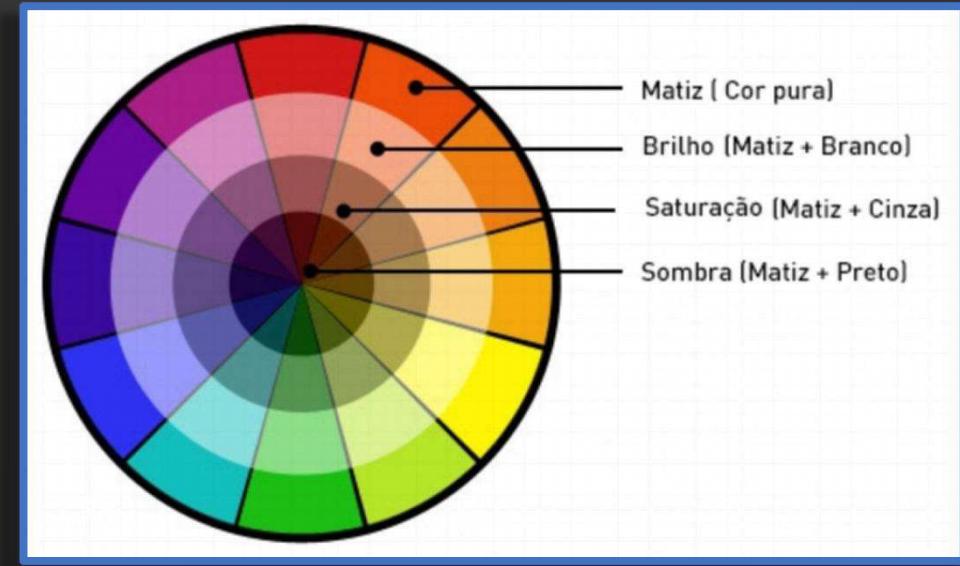
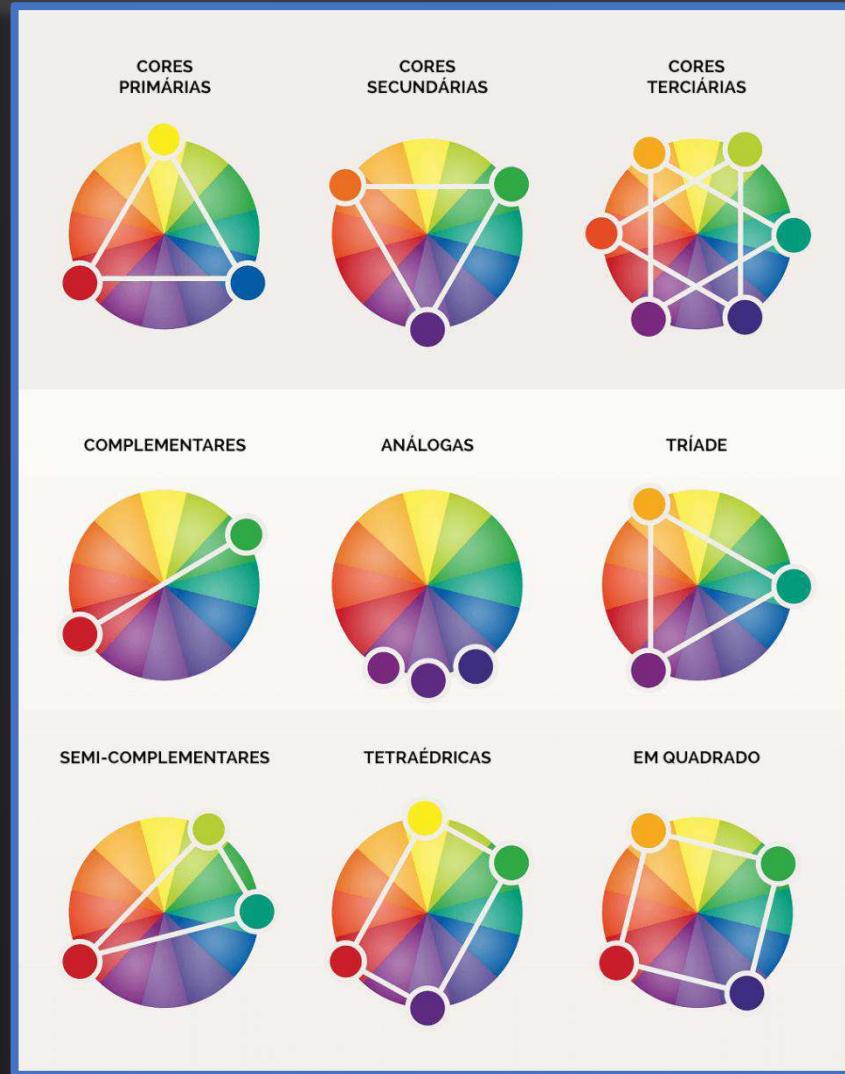
Caso deseje atribuir à um elemento H1, uma cor de texto vermelho, podemos escolher uma das notações disponíveis

<h1> Meu Titulo </h1> → *html*

```
h1 {
    color: red;
ou
    color: #ff0000; → css
ou
    color: rgb(255,0,0);
ou
    color: hsl(0, 100%, 50%);

}
```

Meu Titulo → *resultado*



UNIDADES DE MEDIA

O CSS possui várias unidades diferentes para expressar um comprimento. Estas unidade se dividem em 02 tipos: Absoluta e Relativa

ABSOLUTA - São medidas fixas, ou seja, mesmo que o tamanho da tela varia a medida será a mesma. Ex: **cm** (centímetros), **mm** (milímetros), **in** (polegadas 1in = 96px = 2.54cm), **px** (pixel 1px = 1/96th of 1in), e **pt** (pontos 1pt = 1/72 of 1in)

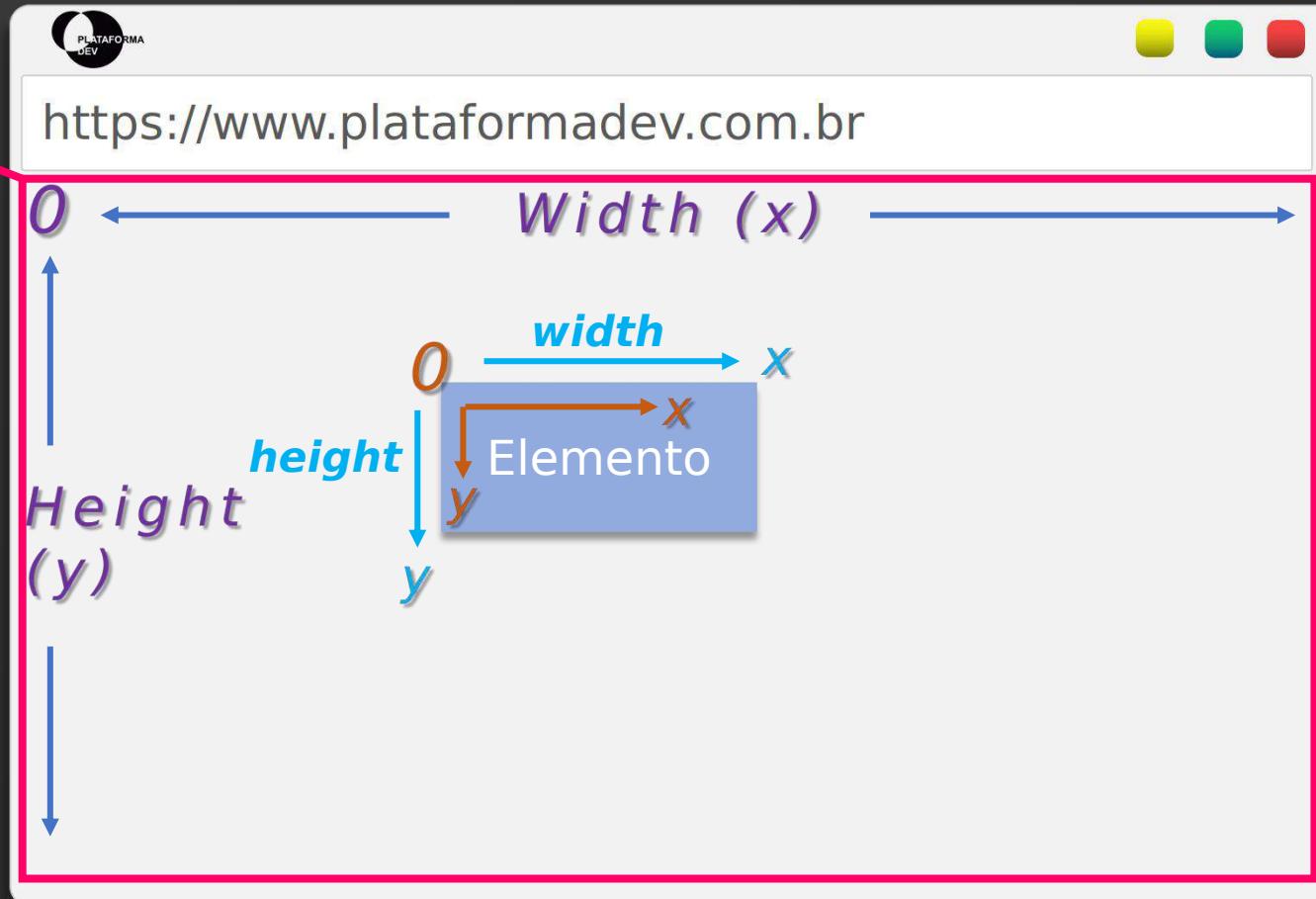
RELATIVA - São medidas relativas a outras medidas de comprimento, Ex: **em** (relativo ao tamanho da fonte do elemento pai), **ex** (relativo à altura X do elemeto atual), **ch** (relativo à largura do número 0), **rem** (relativo ao tamanho da fonte do elemento root), **vw** (relativo à 1% da largura da viewport), **vh** (relativo à 1% da altura da viewport) e **%** (relativo ao elemento pai)

Geralmente se usa medidas absolutas, para elementos que se deseja ter o tamanho fixo, independente do dispositivo em que o cliente esteja usando para visualizar sua aplicação, e unidades relativas para elementos responsivos.

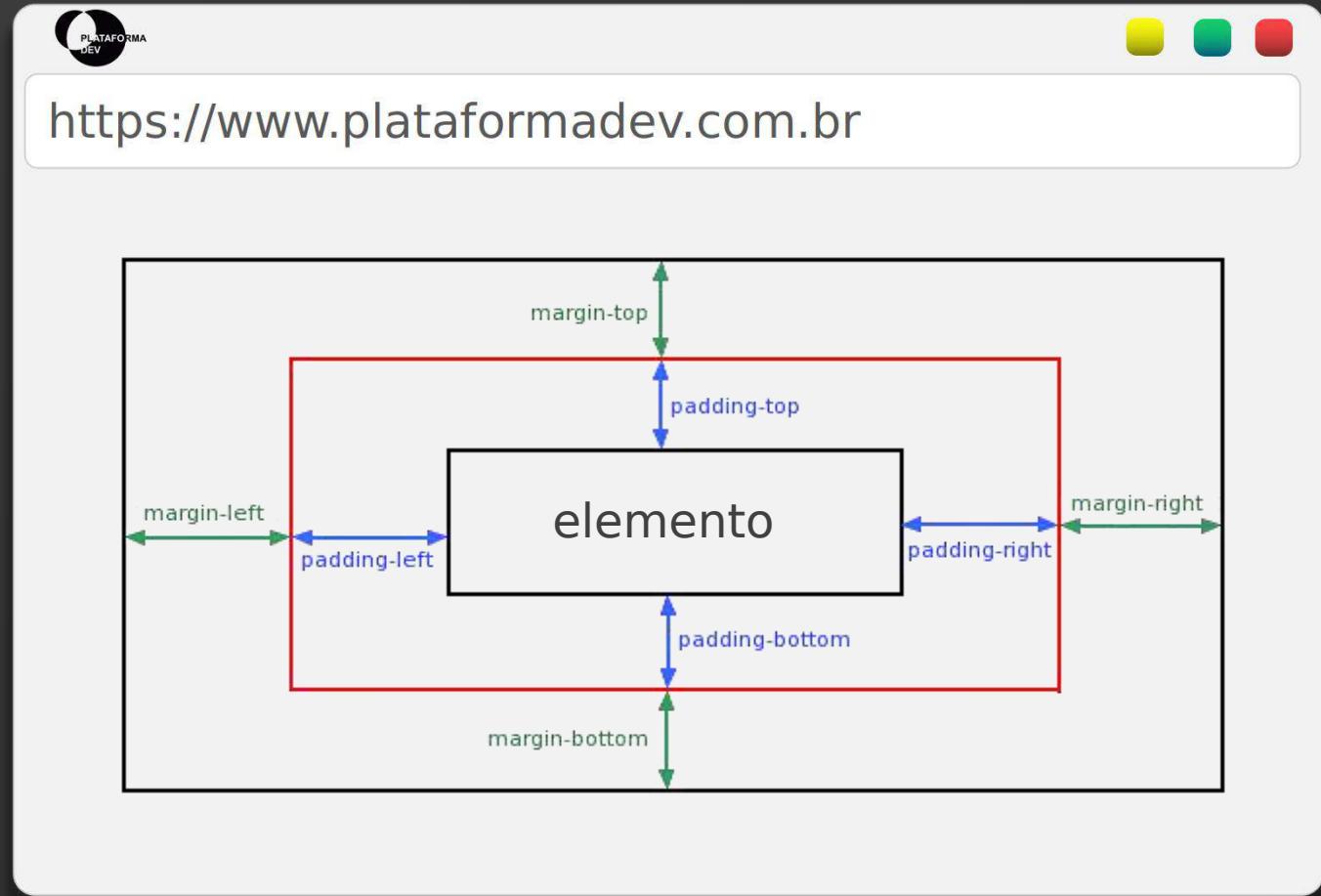
REFERÊNCIA RELATIVA DA PÁGINA

Viewport

Área da tela, que está visível para o cliente, independente de se ter uma tela enorme em altura, esta medida refere-se ao que o cliente está vendo no momento



MARGIN / PADDING / BORDER



UNIDADES DE MEDIDA
USADAS PARA
ADICIONAR VALOR

REM

EX

VW

VH

VMAX

VMIN

CH

PX

MARGIN

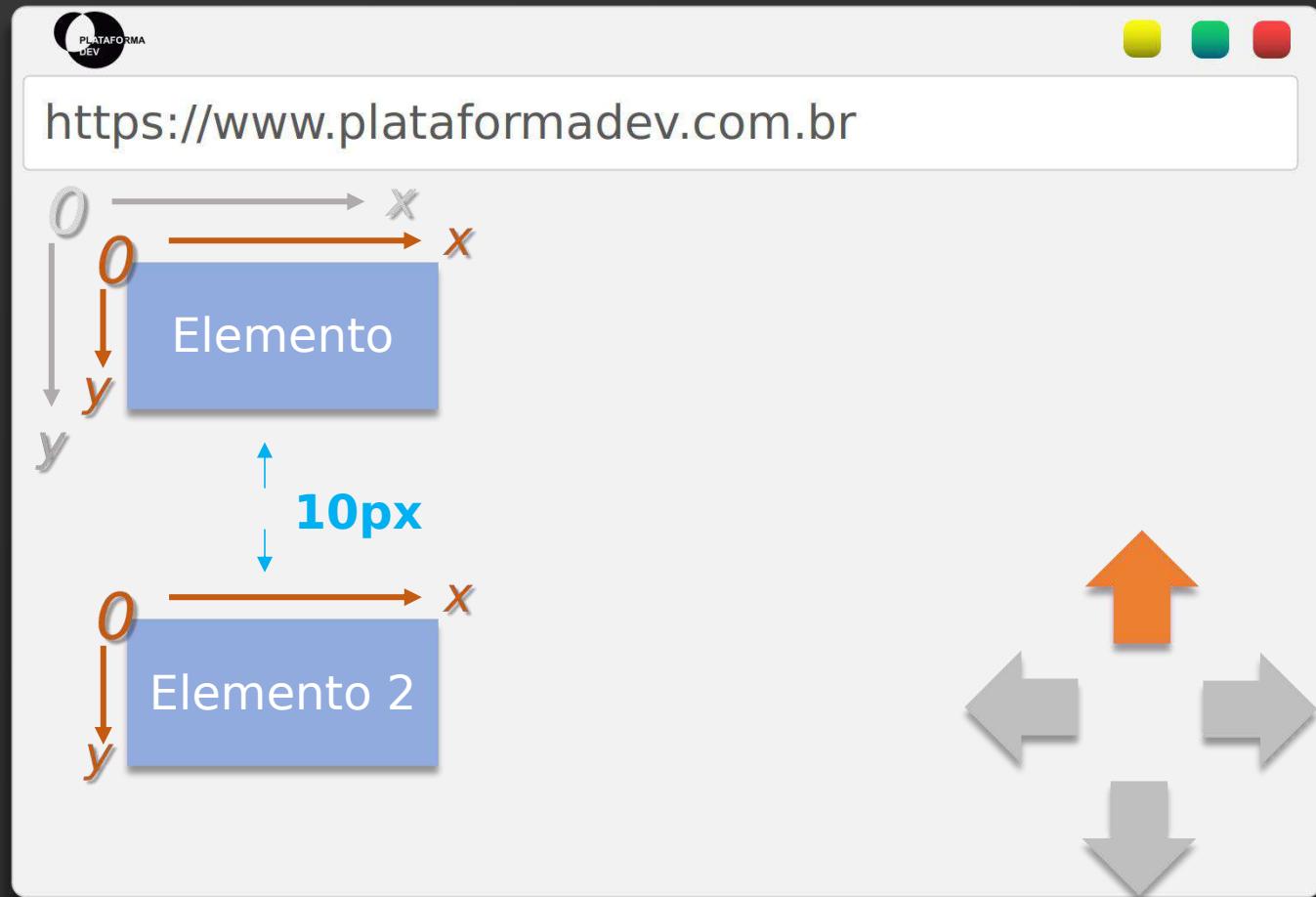
Define o espaço externo do elemento, neste exemplo: espaço entre o topo da página e o Elemento

```
Elemento {  
  margin-top: 10px;  
}
```



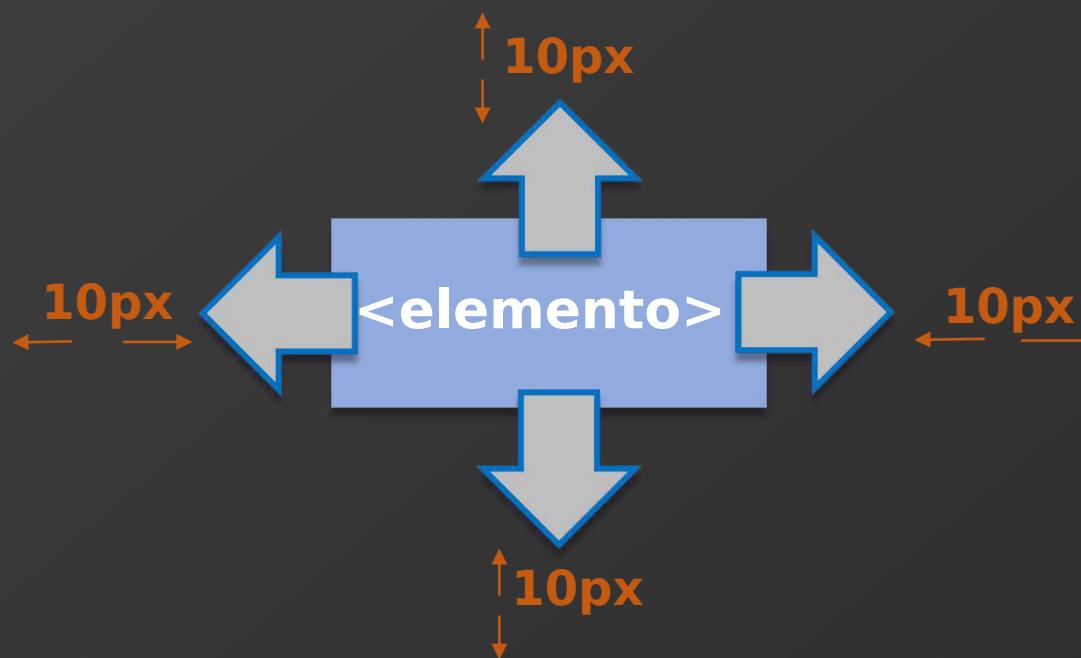
Neste exemplo:
Espaço entre o 1º elemento e o 2º Elemento

```
Elemento2 {  
  margin-top: 10px;  
}
```

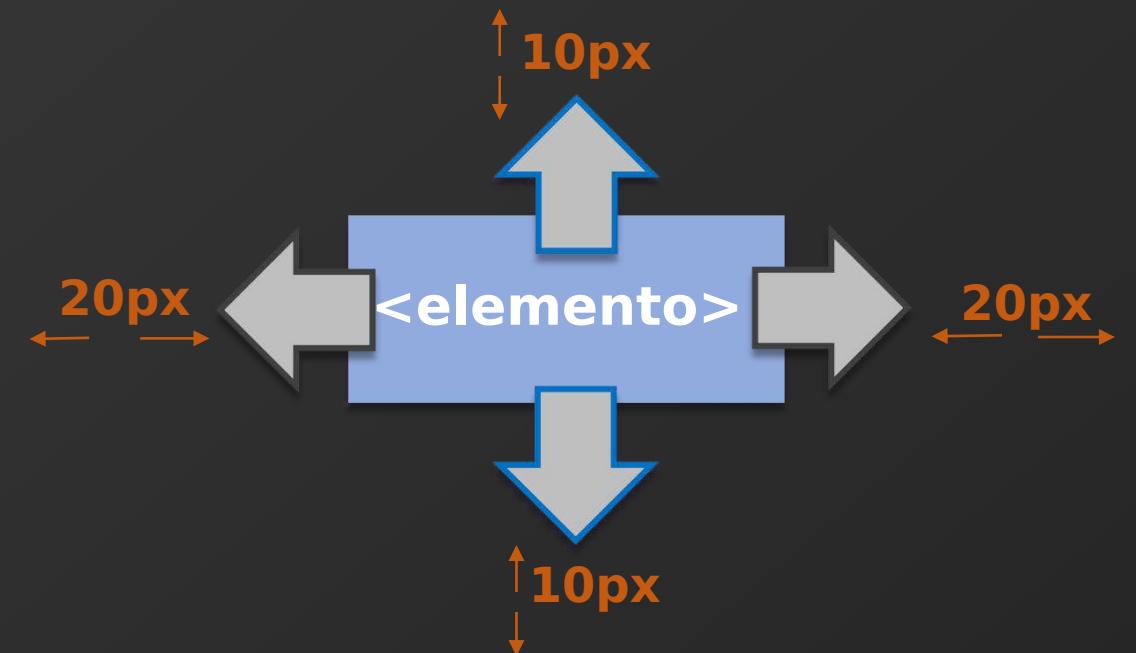


DECLARAÇÕES

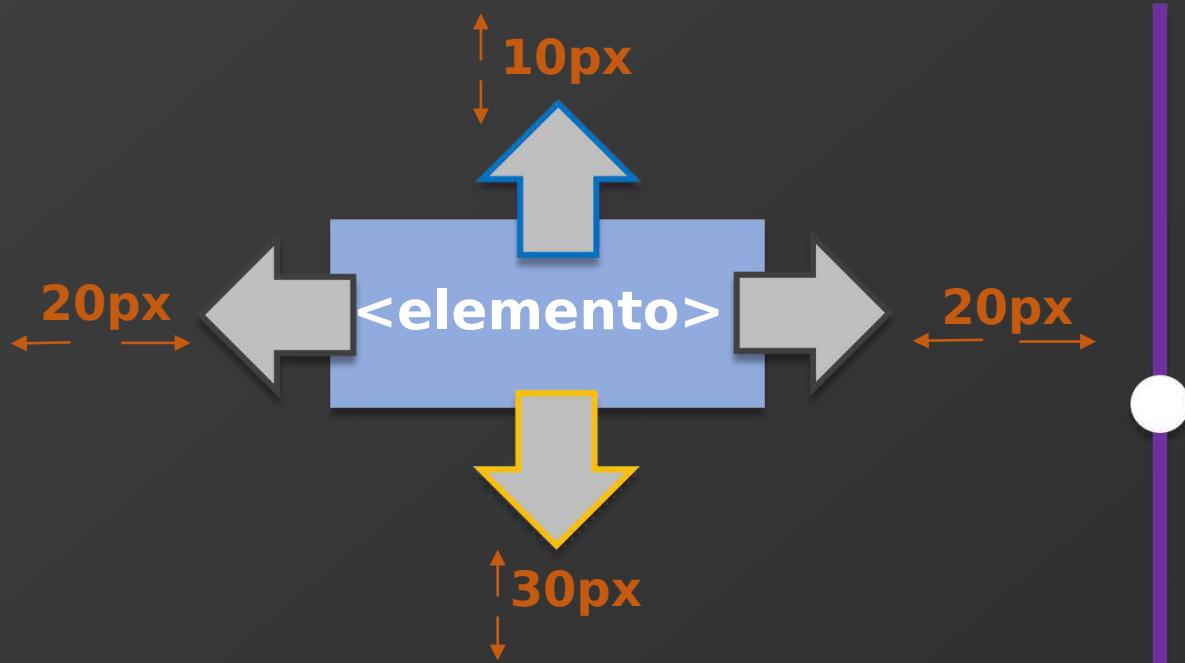
Informa a direção à que a propriedade vai ser aplicada: PADDING, MARGIN ou POSICIONAMENTO



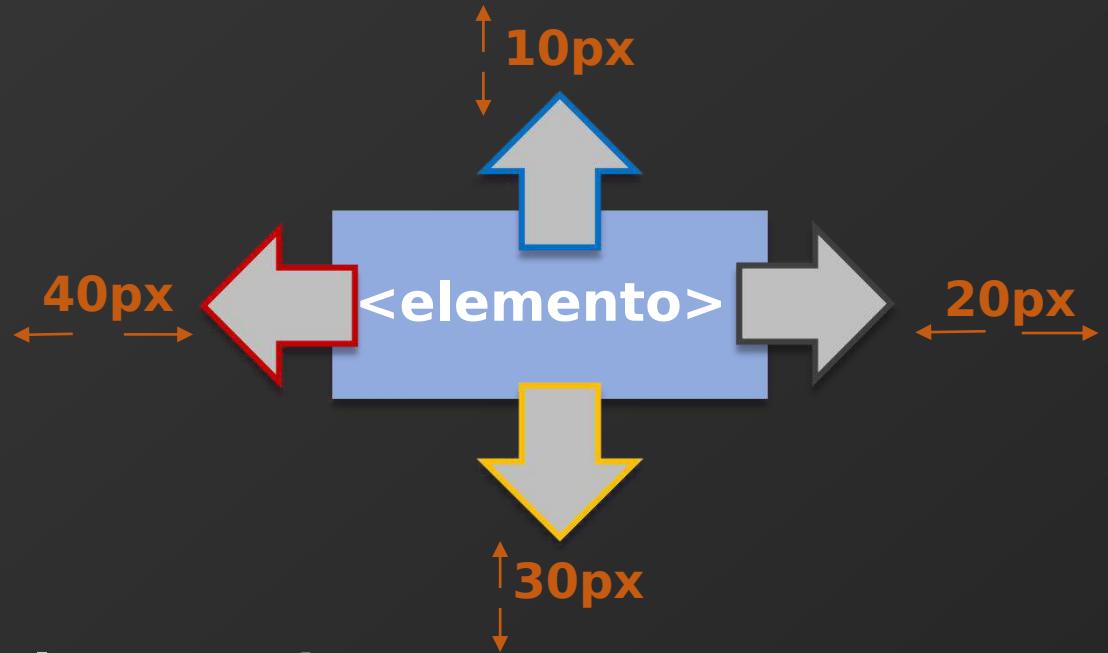
```
elemento{  
    propriedade: 10px;  
}
```



```
elemento{  
    propriedade: 10px 20px;  
}
```



```
elemento{  
    propriedade: 10px 20px 30px;  
}
```



```
elemento{  
    propriedade: 10px 20px 30px 40px;  
}
```

BOX-SIZING

Box-Sizing: Content-Box

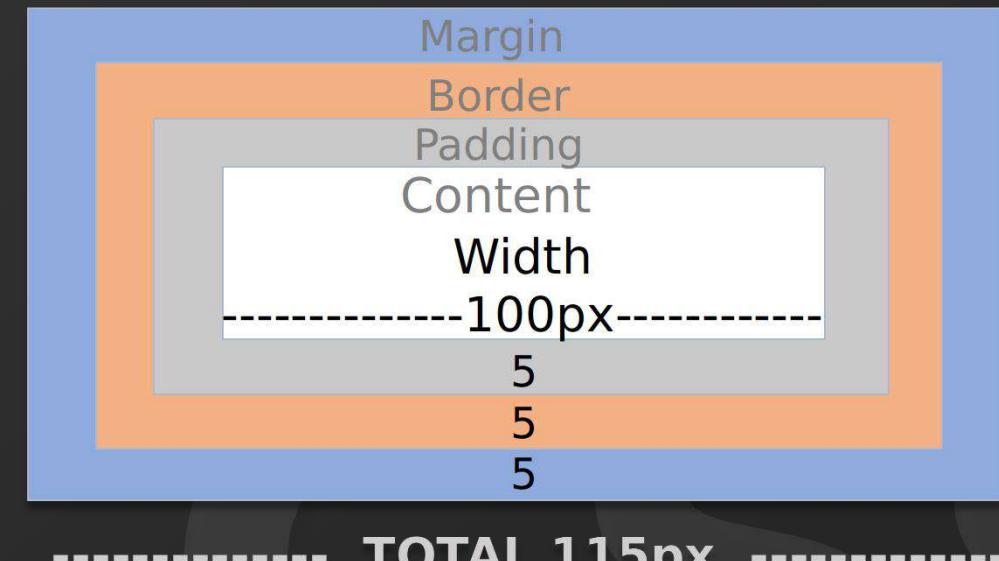


Element {

```
width: 100px;  
margin: 5px;  
padding: 5px;  
border: 5px;  
box-sizing: ??? ;
```

}

Box-Sizing: Border-Box



ESTADOS DO LINK

Os estados em links são personalizados através de declarações de regras de estilo para as pseudo-classes do elemento `<a>` do HTML, veja a sintaxe seguinte.

seletor:pseudo-classe {propriedade: valor}

Sendo que as classes em CSS podem também ser usadas com pseudo- classes.

seletor.class:pseudo-class {propriedade: valor}

Possíveis estados:

a:link.....define o estilo do link no estado inicial;

a:visited...define o estilo do link visitado;

a:hover....define o estilo do link quando passa-se o mouse sobre ele;

a:active....define o estilo do link ativo (o que foi "clicado").

FLEXBOX

A propriedade flexbox deve ser adicionada à um elemento **pai (container)**, desta maneira podemos manipular seus elementos filhos, tanto no **eixo principal(main axis)** quanto no **eixo transversal (cross axis)**, para conseguirmos usá-lo basta que defina no elemento que será o container, a propriedade **display: flex;**

CONTAINER

FLEX-DIRECTION : **row** | row-reverse | column | column-reverse

FLEX-WRAP : **nowrap** | wrap | column | wrap- reverse

JUSTIFY-CONTENT : **flex-start** | flex-end | center | space- between | space-around | space-evenly

ALIGN-ITEMS : **stretch** | flex-start | flex-end | center | baseline

ALIGN-CONTENT : **stretch** | flex-start | flex-end | center | space-between | space-around

ITEM

ALIGN-SELF : stretch | flex-start | flex-end | center | baseline

ORDER : **0** | número

FLEX-GROW : **0** | número

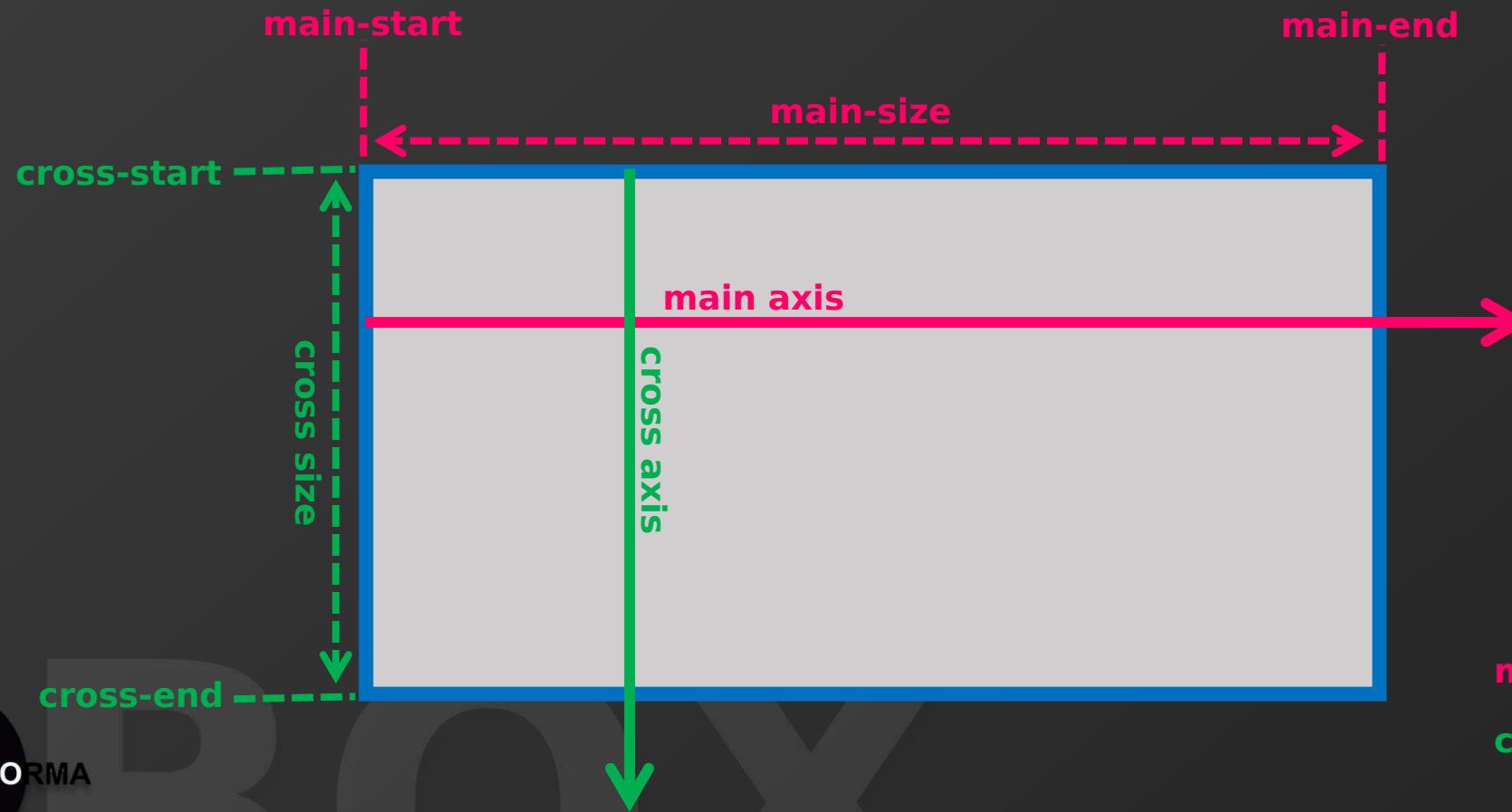
FLEX-SHRINK : **1** | número

FLEX-BASIS : **auto** | length

FLEX : **0 1 auto**

AXIS

Tanto o eixo principal(MAIN AXIS), quanto o eixo transversal(CROSS AXIS) dependem do valor aplicado à propriedade flex-direction, caso o flex-direction seja ROW, o Main Axis é na horizontal como a imagem abaixo, e caso seja o flex-direction seja COLUMN, o Main Axis é na Vertical, ou seja, os eixos se invertem.



main axis = eixo principal

cross axis = eixo transversal

FLEX-DIRECTION

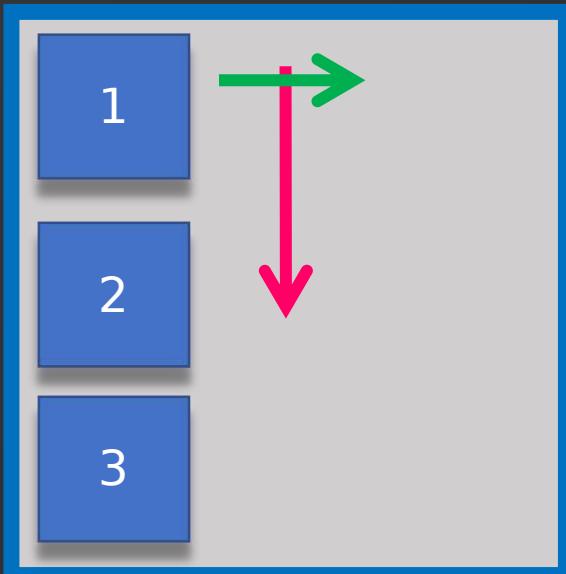
row



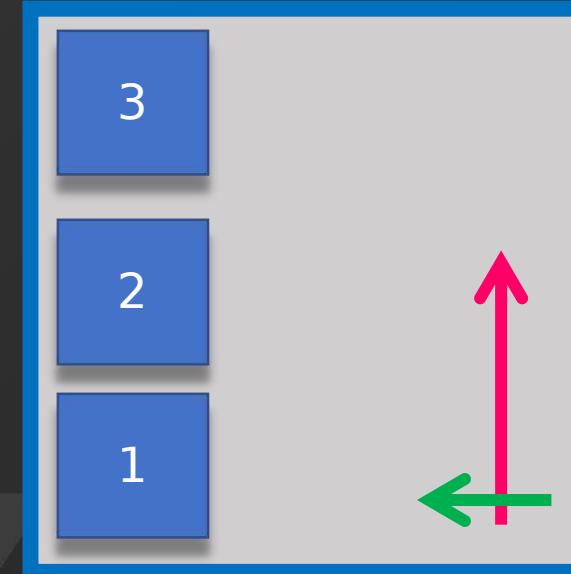
row-reverse



column



column-reverse



- main axis —————
- cross axis —————
- container
- item

JUSTIFY-CONTENT

flex-start



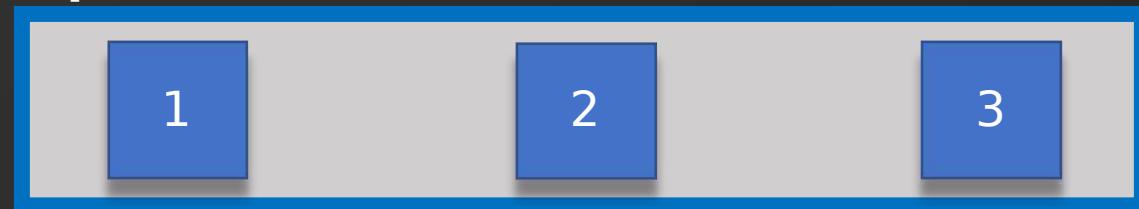
space-between



flex-end



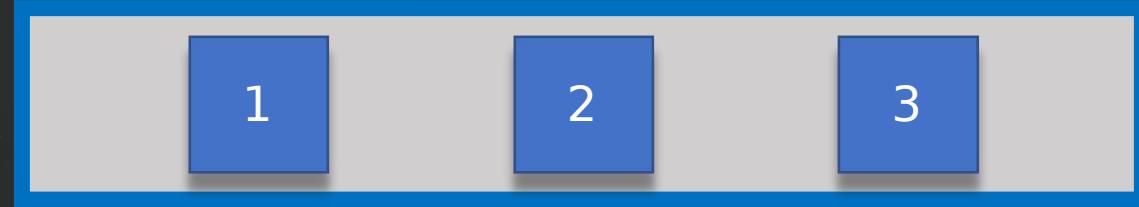
space-around



center

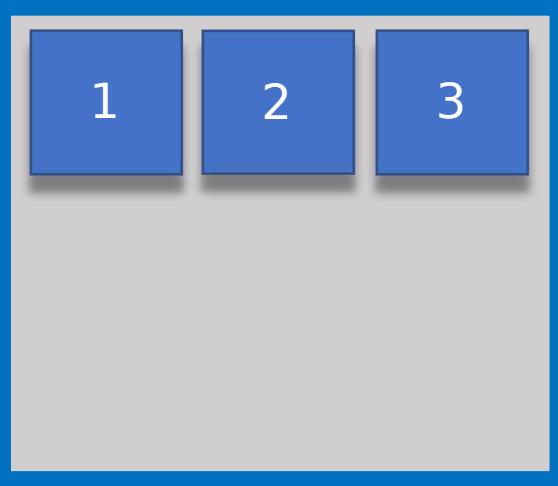


space-evenly

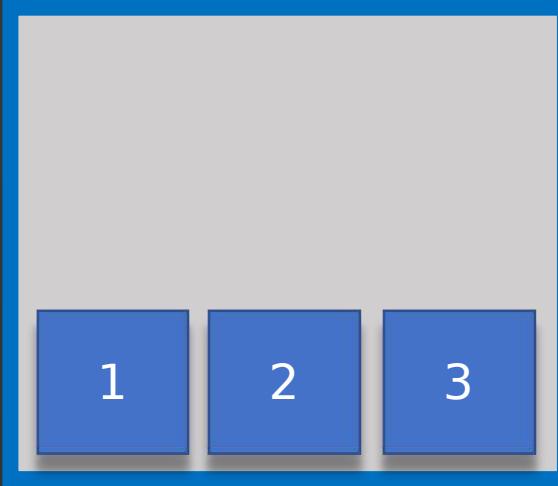


ALIGN-ITEMS

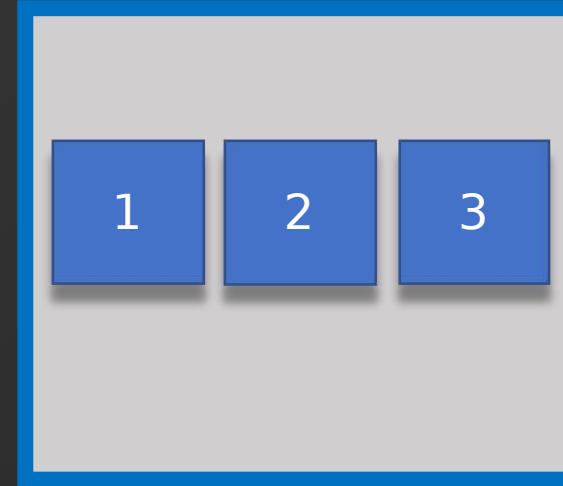
flex-start



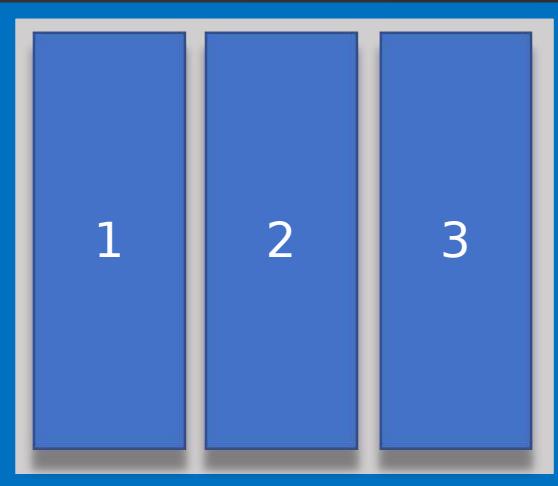
flex-end



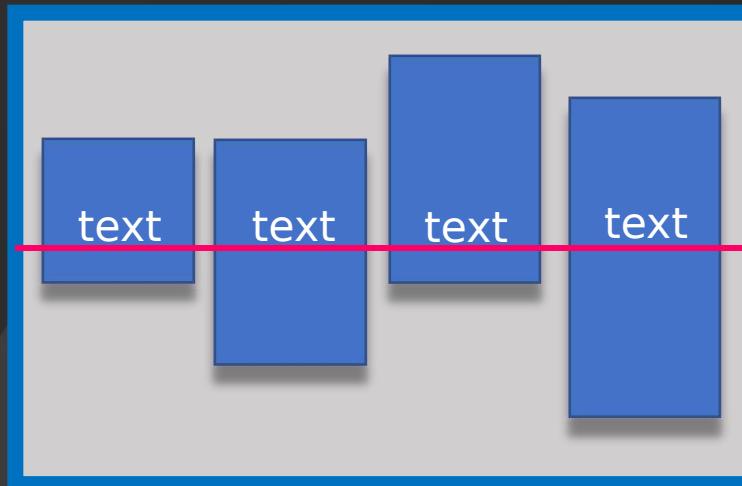
center



stretch

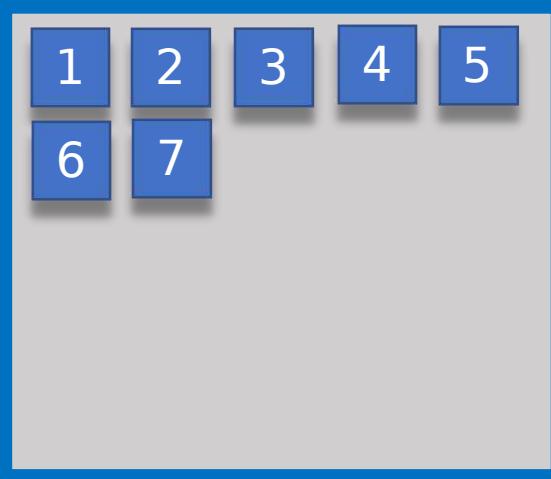


baseline

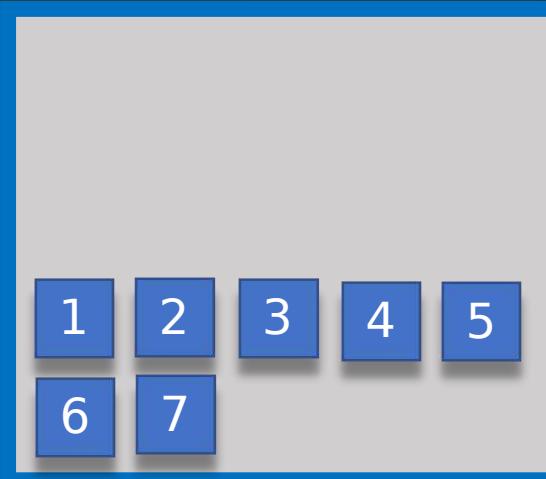


ALIGN-CONTENT

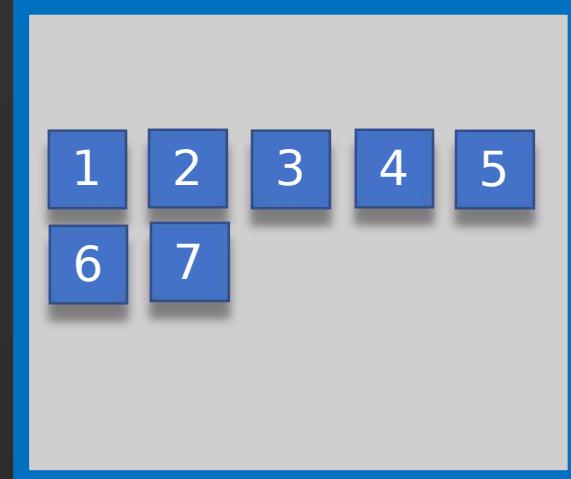
flex-start



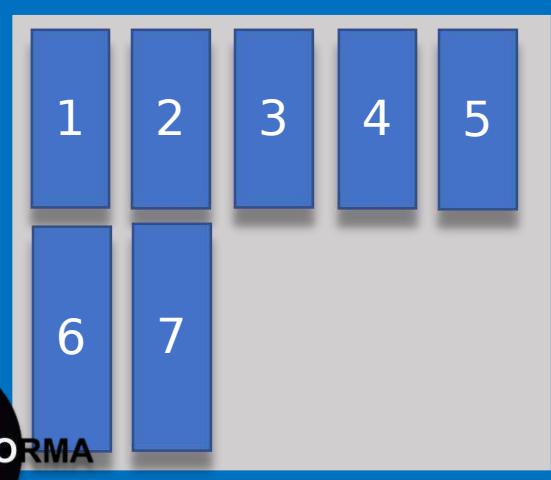
flex-end



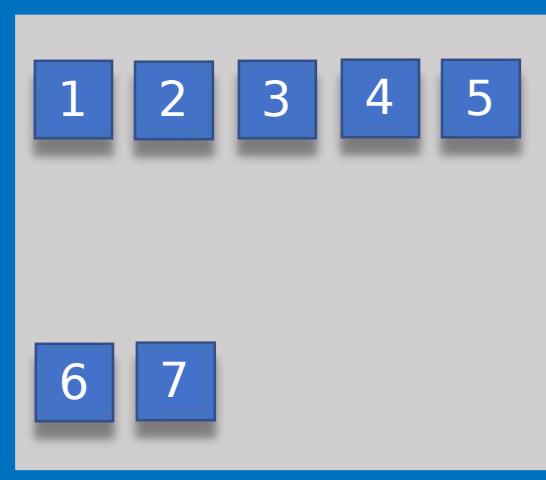
center



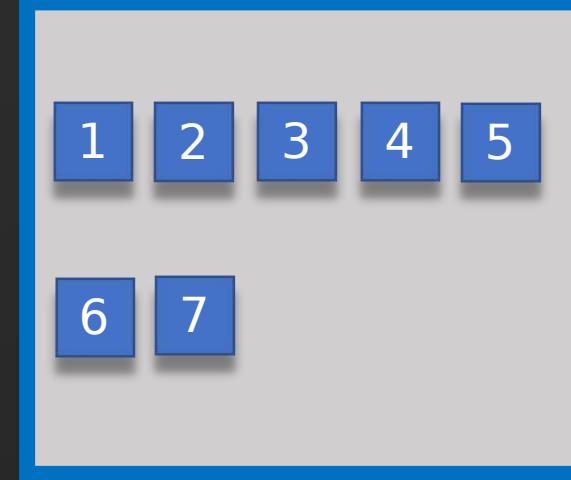
stretch



space-between

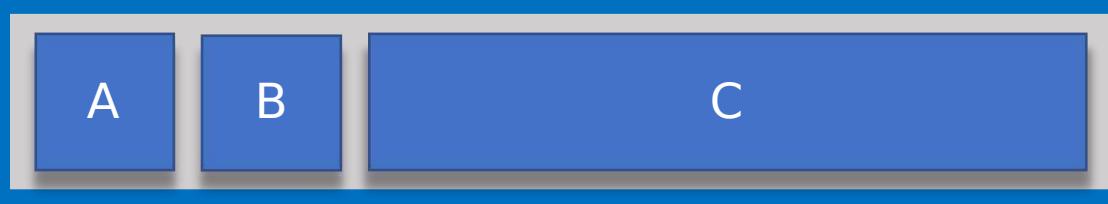


space-around

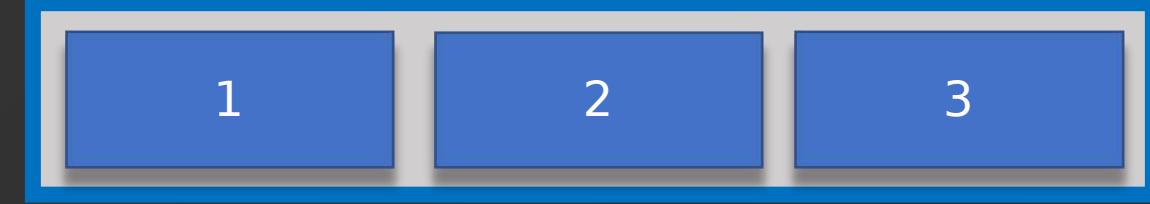


FLEX-GROW & WRAP

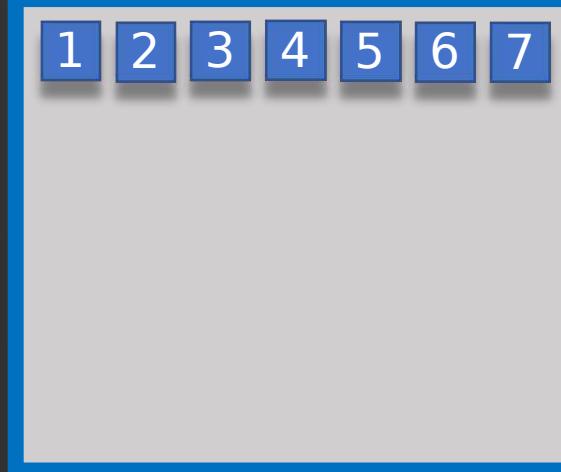
A,B { flex-grow: 1 }
C {flex-grow: 6 }



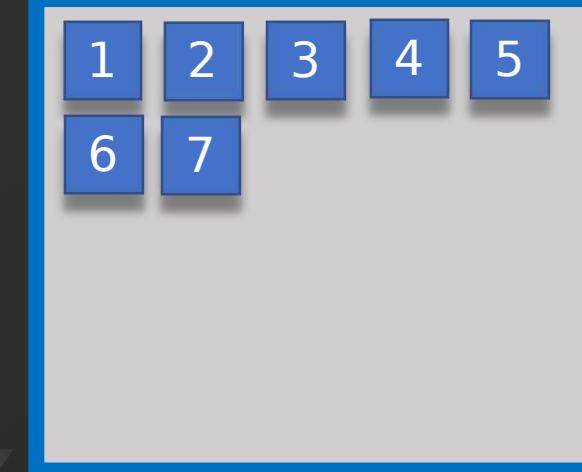
A,B,C{ flex-grow:1 }



nowrap



wrap



Exemplo:

Abaixo temos um exemplo simples que pode ser usado para o alinhamento de um menu, ou de cards, com links, etc.

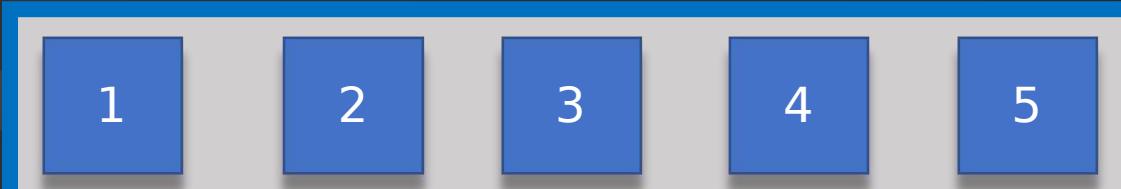
HTML

```
<div class="container">
  <div> 1 </div>
  <div> 2 </div>
  <div> 3 </div>
  <div> 4 </div>
</div>
```

CSS

```
.container {
  display:flex;
  justify-content:space-between;
}
```

Resultado



A **@media** é usada em consultas de mídia para aplicar estilos diferentes para diferentes tipos / dispositivos de mídia.

As consultas de mídia podem ser usadas para verificar várias coisas, como:

- largura e altura da janela de visualização
- largura e altura do dispositivo
- orientação (o tablet / telefone está no modo paisagem ou retrato?)
- resolução

O uso de consultas de mídia é uma técnica que fornece uma folha de estilo personalizada para desktops, laptops, tablets e telefones celulares, fazendo com que o conteúdo da página se adeque a cada tipo de dispositivo usado pelo cliente.

Outro fator importante além da consulta de mídia, é o uso da meta tag viewport da seguinte forma:

```
<meta name="viewport" content="width=device-width" />
```

Isso irá indicar ao navegador que o width da metatag viewport é o tamanho da largura do dispositivo

SINTAXE

```
@media not|only mediatype and (mediafeature and|or|not mediafeature) {  
    CSS-Code;  
}
```

not: a palavra-chave not reverte o significado de uma consulta de mídia inteira.

only: a única palavra-chave impede que navegadores mais antigos que não suportam consultas de mídia com recursos de mídia apliquem os estilos especificados. *Não tem efeito nos navegadores modernos.*

and: A palavra-chave e combina um recurso de mídia com um tipo de mídia ou outros recursos de mídia.

mediatype: os tipos podem ser: All, print, screen, speech

HTML

VISUALIZANDO NO DESKTOP

<div> Example </div>

Example

CSS

```
div {  
    background: green;  
}
```

VISUALIZANDO NO MOBILE

```
@media screen and (max-width: 450px) {  
  
    div {  
        background: yellow;  
    }  
  
}
```

Example



media

— INTRODUÇÃO ao

JAVASCRIPT



JAVASCRIPT

É uma linguagem de programação interpretada e de alto nível, desenvolvida por Brendan Eich na década de 90.

Em 1996 virou um padrão chamado ECMAScript, na qual determina certas regras para os navegadores.



LOCALIZAÇÃO CÓDIGO JS

Código JS sendo chamado em um documento externo

HTML

```
<script src="app.js"></script>
```

JS

```
alert( 'código js' )
```

Código JS dentro do documento HTML

```
<script>  
  
    alert( 'código js' )  
  
</script>
```

Código JS dentro da tag HTML

```
<button onclick="enviarForm()"> Enviar </button>
```

VARIÁVEIS

As Variáveis em JavaScript reservam um espaço na memória do computador (como se fosse uma caixa), onde podemos guardar temporariamente determinado tipo de dado, elas podem ser declaradas usando uma das palavras reservadas **Var** , **Let** ou **Const**.



Escopo Global e Local



Escopo Global, e de Bloco



Escopo Global e valor 'imutável'

JavaScript é uma linguagem fracamente tipada, e caseSensitive. Além de ser multiparadigma. Praticamente tudo em Javascript trata-se de um objeto

Declarar variável

```
let nome
```

Atribuir valor

```
nome = 'Fulano'
```

Usar variável

```
console.log(nome)
```

P R I M I T I V E

String

Number

Boolean

Null

Undefined

Symbol

```
let nome = 'Fulano'
```

```
let idade = 34
```

```
let casado = true
```

```
let varNula = null
```

```
let indefinida
```

```
let sym = Symbol("foo")
```

O B J E C T

Object

Array

Date

Function

```
let pessoa = { nome:"Fulano", idade : 34 }
```

```
let pessoa = [ 'abacate', 'uva', 'pera' ]
```

```
let now = new Date()
```

```
function soma(num){ return num + num }
```

- As variáveis não podem iniciar com números, apenas com letras ou '_'
- Não pode ser utilizado caracter especial como 'ç', '^', '~'
- Não podem ser iguais as palavras reservadas da linguagem
- Por convenção quando usar nomes compostos, usamos camelcase
- Dê preferência por nomes em inglês às variáveis, pensando em globalidade
Ex: *novaVariavel*, *firstName*, etc...

String

Dados em formato de texto, valores declarados dentro de áspas, independente de serem simples ou duplas

```
var nome = "Fulano"
```

Object

Dados em formato de par, chave : valor, podendo ter quantos itens forem necessários, igual um Array

```
var pessoa = {  
    nome: "Fulano",  
    idade: 33  
}
```

Number

Dados em formato de números, valores positivos, negativos, com ponto flutuante

```
var idade = 33
```

Symbol

A Symbol() função retorna um valor do tipo símbolo, com atributos e métodos estáticos, todo valor retornado é exclusivo

```
let sym = Symbol("foo")
```

Function

um conjunto de instruções que executa uma tarefa ou calcula um valor, pode ou não retornar algo, pode ou não possuir parâmetros

```
var soma = function(num){  
    return num + num  
}
```

Boolean

Dados do tipo True ou False

```
var casado = true
```

Undefined

São variáveis apenas declaradas mas sem qualquer tipo de valor atribuído à elas, nem espaços na memória são alocados

```
var indefinida
```

Date

Cria uma instância de Date que representa um único momento no tempo, é baseado no valor de tempo que é o número de milisegundos desde 1º de Janeiro de 1970 (UTC).

```
var now = new Date()
```

Null

Dados nulos, sem valor algum, apenas reserva um local na memória mas não aloca qualquer tipo de valor

```
var nulo = null
```

Array

Dados como se fosse uma lista, e cada item da lista possui um índice, por default, o índice inicia sempre em zero (0)

```
var frutas = ['uva','pera']
```

EXEMPLO PRÁTICO

```
let a = 1
let b = 2
let c = 3
let d = 4.2 //double
let e = false //boolean
let f = 'string' //string

//adição
console.log(a + b) //3
console.log(c + f) //3string

//divisão
console.log(d / b) //2.1
console.log(b / a) //2

//multiplicação
console.log(c * b) //6
console.log(b * e) //0

//diminuição
console.log(d - c) //1.20
console.log(c - a) //2
```

```
let a = 5
let b = 8
let c = 3.5
let d = 'string'
let e = false
let f = 9
let g = '9'

//AND
console.log((b > a) && (c > g)) //false
console.log((g > a) && (c < b)) //true
console.log((a > a) && (f < b)) //false

//OR
console.log((d > a) || (f > b)) //true
console.log((g > a) || (d < g)) //true
console.log((b < b) || (f < g)) //false

//XOR
console.log((b > a) ^ (c > g)) //true 1
console.log((g > a) ^ (c < b)) //false 0
```

```
let a = 5
let b = 8
let c = 3.5
let d = 'string'
let e = false
let f = 9
let g = '9'

//maior que
console.log(b > a) //true
console.log(d > a) //false
console.log(f > a) //true

//menor que
console.log(c > 2) //true
console.log(d > e) //false

//diferente
console.log(b != a) //true
console.log(b != 8) //false

//igual
console.log(f == g) //true
```

OPERADORES

OPERADORES ARITMÉTICOS

operador	Descrição
+	soma
-	subtração
*	multiplicação
**	exponenciação
/	divisão
%	módulo
++	incremento
--	decremento

OPERADORES DE COMPARAÇÃO

operador	Descrição
=	atribuição de valor
==	igualdade de valor
====	igualdade de valor e tipo
!=	diferente
!==	diferente valor e tipo
>	maior que
<	menor que
>=	maior igual
<=	menor igual

OPERADORES LÓGICOS

operador	Descrição
&&	AND
~	NOT
	OR
^	XOR

FUNÇÃO

Funções em JavaScript são blocos de códigos(os blocos são delimitados por chaves `{}`). Uma função é como um procedimento de JavaScript — um conjunto de instruções que executa uma tarefa ou calcula um valor. Para usar uma função, você deve definí-la em algum lugar no escopo do qual você quiser chamá-la,Em Javascript há mais de uma forma de definir funções. Cada forma pode alterar a maneira de como e quando podemos invocá-las e manipulá-las.

Elas podem ser criadas de forma literal.

```
function myFunction(){ } //definindo uma função
```

Passadas como parâmetros para outras funções(callback)

```
myFunction(function(){ console.log("função como parâmetro") })
```

Existem ainda inúmeras outras formas de se trabalhar com funções, no entanto o entendimento básico é como criá-las e como utilizá-las de forma correta no seu código

IF ELSE

```
if ( condição )  
{  
    Bloco Condição Verdadeira  
}  
else  
{  
    Bloco Condição falsa  
}
```

```
if( 5 > 3 ){  
    console.log('true')  
} else {  
    console.log('false')  
}
```

// 'true'

```
if( 1 > 3 ){  
    console.log('true')  
} else {  
    console.log('false')  
}
```

// 'false'

LAÇO FOR

```
for( 01- inicializa  
      contador ; 02-condição ; 04-incremento )  
{  
    03 - comandos  
}  
}
```

```
for(let i=0; i < 3; i++){  
    console.log(i)  
}
```

// 0 1 2



```
Switch ( 02-teste )  
{  
    Case 1:  
        //comandos a serem executados  
        Break;  
  
    Case 2:  
        //comandos a serem executados  
        Break;  
  
    Case 3:  
        //comandos a serem executados  
        Break;  
  
    default  
        //comandos a serem executados  
}
```

```
let fruta = 'uva'  
  
switch( fruta ){  
    case 'pera':  
        console.log( 'peras custam R$2,00' )  
        break;  
  
    case 'maçã':  
        console.log( 'maçã custam R$3,00' )  
        break;  
  
    case 'uva':  
        console.log( 'uvas custam R$5,00' )  
        break;  
  
    default:  
        console.log( 'desculpe não temos' )  
}  
  
// uvas custam R$5,00'
```



LAÇO WHILE

01- inicializa
contador

```
while( 02-condição )  
{  
    03 - comandos  
}  
  
    04-incremento  
}
```

```
let i = 0  
  
while( i < 3 ){  
    console.log( i )  
    i++  
}
```

// 0 1 2

MÉTODOS PARA STRINGS

`const varString = "palavra"`

0	1	2	3	4	5	6
p	a	l	a	v	r	a

`varString.charAt(4)` // v

`varString.concat(' portuguesa')` // palavra portuguesa

`varString.indexOf('r')` // 5

`varString.replace('vra','word')` // palaword

`varString.split('v')` // ['pala', 'ra']

`varString.toUpperCase('v')` // PALAVRA

`varString[5]` // 'r'

MÉTODOS PARA ARRAYS

```
const listFruit = ['pera','uva','maçã']
```

0	1	2
pera	uva	maçã

```
listFruit.pop() // ['pera', 'uva']
```

```
listFruit.push('abacate') // ['pera', 'uva', 'maçã', 'abacate']
```

```
listFruit.indexOf('pera') // 0
```

```
listFruit.reverse() // ['maçã', 'uva', 'pera']
```

```
listFruit.shift() // ['uva', 'maçã']
```

```
listFruit.forEach( function(item, indice){  
    console.log(item)  
} )
```

// Percorre o Array executa uma função a cada item do array ['maçã', 'uva', 'pera']

MÉTODOS PARA DATE

```
const now = new Date()
```

```
console.log(now)
```

```
// Thu Sep 26 2019 10:34:46 GMT-0300 (Horário Padrão de Brasília)
```

```
now.getDate() // pega dia do mês 1 - 31
```

```
now.getDay() // pega dia semana 0 - 6
```

```
now.getMonth() // pega o mês 0 - 11
```

```
now.getFullYear() // pega o ano yyyy
```

```
now.getHours() // pega a hora 0 - 23
```

```
now.getMinutes() // pega os minutos 0 - 59
```

```
now.getSeconds() // pega os segundos 0 - 59
```

MÉTODOS PARA MATH

Math.round(3.7) // retorna o valor arredondado, 4

Math.abs(-3) // retorna o valor positivo, 3

Math.sqrt(81) // retorna a raíz quadrada, 9

Math.ceil(5.2) // retorna o valor arredondado para cima, 9

Math.floor(4.9) // retorna o valor arredondado para baixo, 4

Math.PI() // retorna o valor da constante PI 3.14

Math.random() //retorna um número randômico entre 0 e 1

Math.pow(6,2) //retorna o primeiro valor elevado à potência do segundo valor, 36

Math.min(6, 2, 9) //retorna o menor valor, 2

Math.max(6, 2, 9) //retorna o maior valor, 9

Document Object Model

- API criada pelo browser
- Gera uma árvore de objetos, através dos elementos criados no HTML
- Facilita a manipulação dos elementos HTML

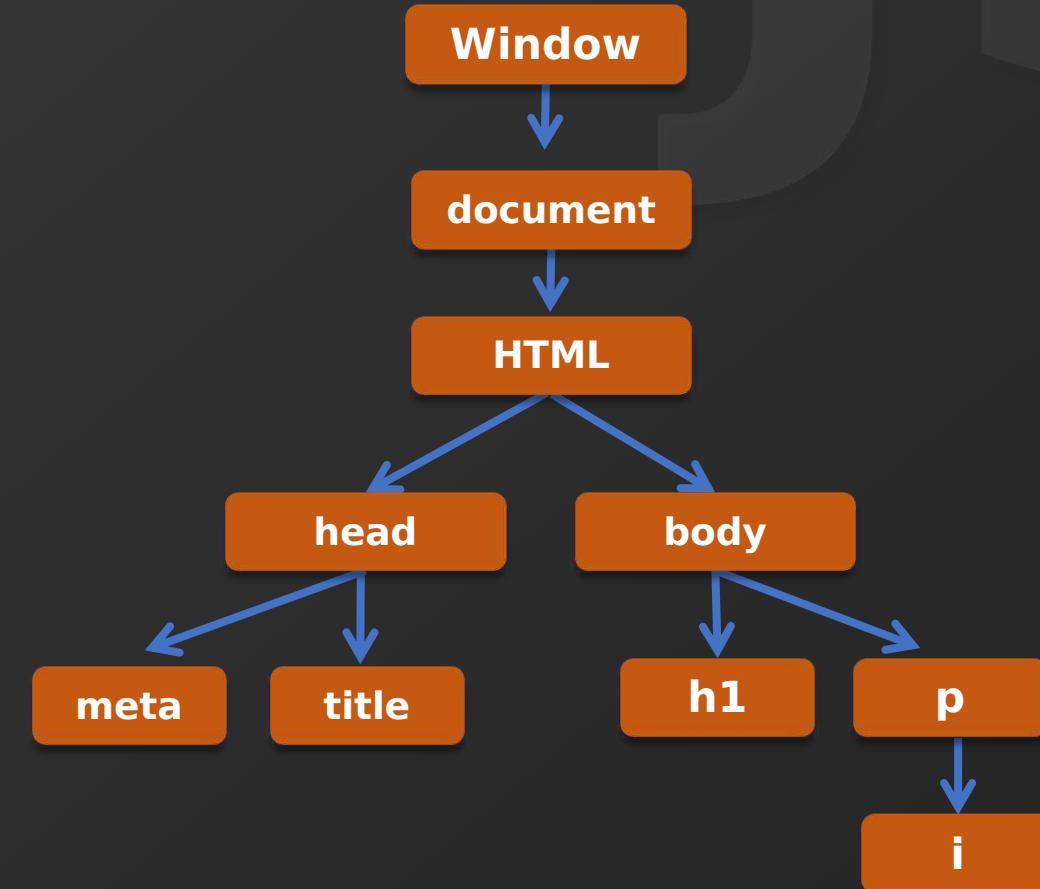
DOCUMENTO HTML / DOM GERADO

Editor de Texto / IDE

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="utf-8">
  <title> Título da página </title>
</head>
<body>

  <h1> Titulo </h1>
  <p>
    Texto de <i> parágrafo </i>
  </p>

</body>
</html>
```



MÉTODOS DE ACESSO AO DOM

Editor de Texto / IDE

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="utf-8">
  <title> Título da página </title>
</head>
<body>

  <h1 id="titulo"> Titulo </h1>

  <h2 class="sub"> Subtitulo </h2>

  <p>
    Texto de <i> parágrafo </i>
  </p>

  <input type="text" name="nome">

</body>
</html>
```

getElementById('titulo')

- Retorna o elemento selecionado através do ID, todo ID se converte em um elemento no window

getElementsByClassName('sub')

- Retorna todos os elementos selecionados através da Class

getElementsByTagName('p')

- Retorna todos os elementos selecionados através da TAG

querySelector('p i')

- Retorna o primeiro elemento selecionado da mesma forma que os selecionamos no CSS, logo aqui seleciono o elemento I que está dentro do elemento P

EVENTOS

Os eventos são basicamente um conjunto de ações que são realizadas em um determinado elemento da página web, seja ele um texto, uma imagem, ou uma div, etc. Os eventos podem ser aplicados de forma inline, diretamente na tag do elemento, ou em arquivo separado, selecionando o elemento no DOM e adicionando um escutador de evento (**addEventListener**).

OnBlur - Remove o foco do elemento

OnChange - Muda o valor do elemento

OnClick - Quando o elemento é clicado pelo usuário

OnFocus - Quando o elemento é focado

OnKeyPress - Quando o usuário pressiona uma tecla sobre o elemento

OnMouseOver - Quando o usuário passa o mouse sobre o elemento

OnSubmit - Define uma ação quando o usuário envia um formulário

```
alvo.addEventListener('tipoEvento' function(evento){  
    console.log('você usou o ', evento)  
})
```

addEventListener é a maneira de registrar uma espera de evento como especificada no W3C DOM.

Seus benefícios são os seguintes:

Permite mais de um manipulador por evento. Isso é particularmente útil em bibliotecas DHTML ou em extensões Mozilla que precisam trabalhar bem mesmo com outras bibliotecas/extensões sendo usadas.

Te dá um pente-fino do estágio em que a espera de evento é ativada (captura ou borbulha).

Funciona em qualquer elemento DOM, não só para elementos HTML.



www.plataformadev.com.br



*Developer by:
RAFAEL ANGRIZANI*

— INTRODUÇÃO ao

GIT &

GITHUB



GIT

Git é um sistema de controle de versões distribuído (SCV), usado principalmente no desenvolvimento de software, mas pode ser usado para registrar o histórico de edições de qualquer tipo de arquivo.

Foi criado por Linus Torvalds em 2005

Linus Torvalds



GITHUB

GitHub é uma plataforma de hospedagem de código-fonte, com controle de versão que usa o Git. Ele permite que programadores, contribuam em projetos privados e/ou Open Source de qualquer lugar do mundo.

O GitHub foi desenvolvido por Chris Wanstrath, J. Hyett, Tom Preston-Werner e Scott Chacon usando Ruby on Rails, em fevereiro de 2008, em 2018 a Microsoft se tornou a proprietária do Github.



Chris Wanstrath



J. Hyett



Scott Chacon



Tom Preston-Werner

CONFIGURAÇÃO INICIAL DO GIT

1 - Passo

Instalar o git na sua máquina de acordo com o seu sistema operacional, para isto basta seguir a documentação disposta no site do Git.

Linux / Mac: <https://git-scm.com/>

Windows: <https://gitforwindows.org/>

2 - Passo

Após feita a instalação corretamente, você pode checar a versão instalada usando o comando `git --version`. Após a instalação, precisaremos realizar algumas configurações iniciais em nosso Git, e para isto usamos os seguintes comandos:

git config --global user.name “Fulano de Tal”

Este comando configurará globalmente o seu nome de usuário no Git

git config --global user.email “Fulanodetal@email.com”

Este comando configurará globalmente o seu email no Git

git config --list

A partir deste comando você terá acesso a todas as configurações do seu Git, ou você pode usar: `git config user.name` para ver nome do usuário, ou `git config user.email` para ver o e-mail

CONFIGURAÇÃO INICIAL DO GITHUB

Basicamente a configuração inicial do Github, é apenas um cadastro que você precisa realizar, inserindo um **E-mail** e uma **Senha**.

Dentre as configurações após a sua conta criada, você pode escolher uma foto de perfil, um username, dentre outros, para ter acesso as configurações, basta que você procure no menu localizado na parte superior do site com uma seta (menu dropdown), e clique em Settings ou Configurações.

Após estas configurações iniciais tanto do Git quanto do Github, você pode retornar ao terminal e realizar as configurações de Chave pública e privada, caso ainda não as tenha feito, pois estas serão necessárias posteriormente, para que possamos subir nossos projetos ao Github.

CHAVE SSH

O SSH é um protocolo que serve para autenticar um usuário remoto à um servidor, através dele geramos duas chaves SSH, uma pública e uma privada, a chave pública será enviada ao Github, e a chave privada ficará contida em nosso computador, desta forma conseguiremos abrir a chave pública enviada ao github (novo repositório remoto). É também, através destas chaves, que o github consegue identificar quem está subindo os arquivos. As chaves estão localizadas em: ~(home) na pasta **/.ssh**, para termos acesso usamos:

cd ~/.ssh

Este comando fará você navegar até o local . Entretanto caso apareça a mensagem que, não é possível acessar, ou que não exista, ou o diretório não foi encontrado, isto significará que nunca foram geradas chaves, neste ambiente local. Então o próximo passo será gerá-las. Se as tiver pule alguns dos passos seguintes.

GERAR CHAVES / CRIAR SENHA

ssh-keygen -C “seu e-mail do github”

Este comando irá criar um par de chaves: uma pública e uma privada, neste passo lhe será perguntado o caminho completo, onde será salva a pasta **.ssh** e o arquivo **id_rsa** (*chave privada*), por padrão deixamos como está, então pressionamos enter e será criado a pasta e o arquivo, no diretório padrão.

Em seguida será solicitado que você digite uma *senha para esta chave id_rsa*, (guarde esta senha), e que a repita posteriormente. Se não quiser criar a senha, apenas clique Enter nas duas vezes

Após, serão criados os dois arquivos com nossas chaves, na pasta oculta **.ssh**

id_rsa = arquivo com chave privada, **id_rsa.pub** = arquivo com chave pública

ACESSAR AS CHAVES

O primeiro passo agora que temos as chaves criadas, será acessar a pasta com os nossos arquivos com as chaves, ~(home) pasta `/.ssh`, então: `cd ~/.ssh`, assim que acessarmos, usamos o comando `ls`, para listarmos seu conteúdo, desta forma temos: `id_rsa` sendo nossa chave privada e `id_rsa.pub` como sendo nossa chave pública.

COPIAR CHAVE PÚBLICA

Para isto usamos o comando `cat <nome do arquivo>`, este comando mostrará o conteúdo do arquivo escolhido (`cat id_rsa.pub`), neste caso mostrá a nossa chave, que mais parece um aglomerado de números e letras, então copiamos todo o seu conteúdo. Você pode também abrir este arquivo usando o seu editor de texto preferido, o que importa é copiar o conteúdo da chave, para levá-la até o Github.

ENVIAR CHAVE PÚBLICA AO GITHUB

Acessando o site do Github, e estando logado em sua conta, vá até **Settings** (configurações), e no menu lateral clique em **SSH and GPG Keys**, lhe será mostrado as chaves que podem ser adicionadas, clique no botão **New SSH Key**, clicando no botão aparecerão 02 campos, **Title** (para nomear / identificar a chave), e **Key** (que é o local onde você deve por o conteúdo copiado da chave). Em seguida clique no botão **Add SSH Key** para que a chave seja adicionada.

CONECTAR AS CHAVES / GITHUB

Após adicionarmos a chave, no terminal usamos o comando: `ssh -T git@github.com`, lhe será mostrado a mensagem de que a autenticidade não foi estabelecida, e se queremos continuar com a conexão, confirmamos (yes), ele nos avisa que vai adicionar permanentemente à lista de host conhecidos, e que para isto precisamos digitar a nossa senha da chave privada, caso tenha criado uma, clicamos em **desbloquear**, e será adicionado, em seguida nos avisa que nosso usuário está autenticado corretamente, mas não podemos acessar por um terminal SSH. Pronto

ARMAZENAR DADOS

WORKING DIRECTORY (1)

São os arquivos e diretórios que você está usando atualmente como repositório em seu ambiente local

STAGING AREA (2)

É um espaço temporário onde você determina quais mudanças serão adicionadas. Basicamente, você adiciona os arquivos para o staging, e quando estiver satisfeito com as alterações efetua um *commit* a fim de enviar os arquivos ao Datastore

DATASTORE (3)

“Banco de dados” (*Localrepo*) onde o controle de versão decide a forma de empacotamento e guarda as suas alterações

ESTADOS DO

UNTRACKED (1)

GIT

Não Marcado, ou seja, o arquivo foi adicionado ao repositório, mas não foi visto pelo Git, não foi alterado

MODIFIED (2)

Significa “modificado”, ou seja o Git visualizou que você alterou o arquivo, mas ainda não o gravou (*committed*), o arquivo encontra-se no Working Directory, ele deve ser enviado ao Staging area

STAGED (3)

Significa que um arquivo foi modificado e suas modificações foram adicionadas à uma área temporária (staging area)

COMMITTED (4)

Significa “gravado”, ou seja o dado foi armazenado com segurança, deve ser feito a cada alteração significativa (*commit*), o arquivo foi é enviado para o Datastore

Remote

Local

Untracked

working
directory

staging
area

local repo

remote
repo

git add

git commit

git push

git fetch

git checkout

git merge

INICIAR REPOS. GIT

`git init`

Inicializará o repositório git na pasta a ser monitorada, este comando irá criar uma pasta oculta chamada `.git`, que irá monitorar e salvar as informações de versão do projeto que você realizar neste diretório, como é uma pasta oculta, para enxergá-la através do terminal use o comando `ls -a`

APAGAR REPOS. GIT

`rm -rf .git`

Apaga um repositório git que tenha sido inicializado, apagará a pasta oculta `.git`, desde que você o execute no diretório que está sendo usado como repositório, e que você deseja apagar

CHECAR STATUS DO REPOS.

`git status`

Checa o estado atual do repositório, ele avisa se existe ou não modificações para serem gravadas, caso haja alterações nos arquivos monitorados, ele solicitará que os envie ào Staging, usando `git add`

ADICIONAR REPOS. AO STAGING

`git add <nome do arquivo>` OU `git add .`

Adiciona o arquivo a qual foi escrito o nome à staging area. O comando `git add .` adicionará todos os arquivos presentes na pasta atual ao Staging.

REMOVER REPOS. DO STAGING

`git rm --checked <nome do arquivo>` OU `git reset <nome arquivo>`

Retira o arquivo do Staging, caso deseje não monitorá-lo escreva o nome do arquivo dentro de um arquivo chamado `.gitignore` (ficará oculto, caso não exista crie) e adicione o `.gitignore` ao Staging.

COMMITAR REPOS. GIT

git commit -m "Mensagem"

Salva as alterações feitas no arquivo e que estão no Staging ao “banco de dados do GIT”, ele cria uma imagem destas modificações (snapshot), e a flag **-m** é uma mensagem do que foi modificado no arquivo, ao realizar o commit, é gerado ainda um número único, uma hash, que identifica este commit, esta flag -m pode ser usada em conjunto com -a ficando **-am** que significa todos os arquivos alterados

REVERTER COMMIT / ALTERAÇÃO

git revert <hash do commit> OU **git checkout --<nome do arquivo>**

O *git revert*, reverte um commit efetuado em um arquivo específico, em seguida lhe pode ser perguntado qual texto substituirá o texto original, o *git checkout* faz o arquivo escolhido voltar ao seu estado original desde o último commit, ou seja, ele desfaz a última alteração naquele arquivo

HISTÓRICO DE COMMITS

git log OU **git log --pretty=oneline**

O comando *git log* apresenta o histórico de commits realizados no arquivo de forma detalhada, ao usar com a flag **--pretty** ele mostra os commits apenas em uma linha

BUSCA COMMITS POR AUTOR

git log --author="Fulano"

Este comando buscará dentre os commits realizados, aqueles que foram efetuados pelo author especificado na busca

VISUALIZAR 02 ÚLTIMOS COMMITS

git show OU **git show <hash do commit>**

O *git show* apresenta as duas últimas mudanças comitadas, já o comando *git show <hash>*, mostra detalhes específicos daquele commit ao qual a hash está associado

VISUALIZAR NÚMERO DE CONTRIBUIÇÕES

`git shortlog`

Este comando retorna em ordem alfabética os Autores, quantos commits fizeram, e quais foram eles, Você pode usar também o comando `git shortlog -sn`, que retorna o número de commits e quem o fez

VISUALIZAR MUDANÇA ANTES DE COMMITAR

`git diff`

Este comando mostra um log com todas as mudanças feitas no arquivo, a fim de conferência antes de você commitá-lo. Você pode usar o comando `git diff --name-only` e mostrará apenas o nome dos arquivos alterados

COPIAR UM REPOSITÓRIO

`git clone < url do projeto com final .git >`

Clona/Baixa um projeto do GitHub, para isto é necessário a url do projeto no GitHub do qual se deseja fazer o download

FRASES DEFAULT

use "git add <file>" to update what will be committed

use "git add <file>" para atualizar o que será confirmado

-- *Ele está solicitando que você use o comando git add para enviar o arquivo modificado até a staging area para depois ser comitado*

nothing to commit, working directory clean

nada a confirmar, diretório de trabalho limpo

-- *Significa que não possui nada para ser enviado, o seu ambiente de trabalho está limpo*

changes not staged for commit

mudanças não preparadas para confirmação

-- *Elas não foram o staging area para serem comitadas*

no changes added to commit

nenhuma alteração adicionada para confirmar

-- *Significa que você não alterou nenhum arquivo*

on branch master

no mestre da filial ou

no ramo master local atual do Head do Git

head is now at ...

cabeça está agora em ...

-- *Significa em qual versão do arquivo você está trabalhando*

SNAP SHOT

Snapshot é um termo usado, para referenciar todos os estados dos commits realizados, ou seja, quando você realiza um commit é como se ele tirasse uma foto do estado atual do arquivo, lembrando que, para você poder ter acesso aos commits e seus detalhes, como a hash, autor, data, etc, basta usar o comando *git log*.

Então como posso acessar à uma versão específica??

Basta você usar a Hash do commit que você deseja acessar com o comando *git checkout*.

git checkout <hash do commit>

Este comando é usado para acessar determinada versão usando o Hash (identificador único do commit) caso deseje, pode-se usar apenas os 10 primeiros números

A mensagem a seguir mostrará em qual arquivo você se encontra, pois o **HEAD** aponta para a Branch(ramificação) atual em que você está

(HEAD -> master) OU **Head is now at 453klj454 “message commit”**

E caso deseje voltar à última sessão, atualização novamente ??

Basta usar o seguinte comando:

git checkout master

Este comando retornará ao Branch Master (ramificação principal)

ENVIAR PROJETO AO GITHUB

1 - Passo

Para enviarmos um projeto da nossa máquina, para o Github, inicialmente devemos criar um repositório no Github, para receber o nosso projeto, ao realizarmos este passo podemos:

- A) criar os arquivos deste repositório
- B) Empurrar (*push*) algum repositório que temos em nossa máquina para ele
- C) Importar os códigos de outro repositório para ele.

O próprio Github fornecerá os comandos a serem usados para o envio, por ex:

git remote add origin <url do repos. github criado com extensao .git>

Este comando basicamente diz: “git adicione pra mim, um local remoto (repos. github criado por nós), e chame este local de origin”, agora o Git já conhece o local que vai subir o nosso projeto, caso não tenha .git no final coloque. OBS: Caso não saiba qual é o arquivo remoto basta usar o comando *git remote -v* e lhe será mostrado.

3 - Passo

O próximo passo será enviarmos nosso projeto para o repositório Github. para isso usamos:

git push -u origin master

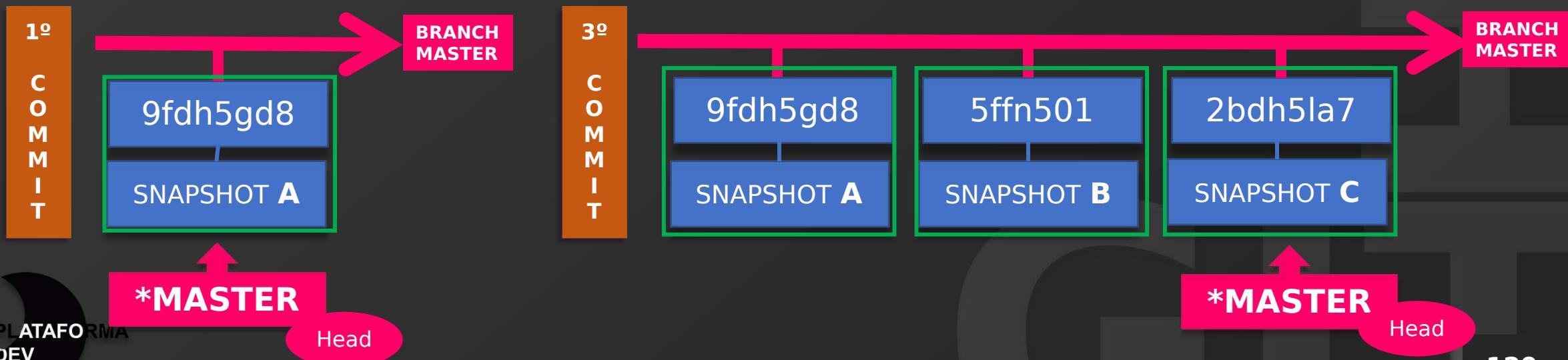
Este comando basicamente diz: “git empurre (push) pra mim, através do usuário local (-u desde que o usuário do git seja o mesmo do github, caso contrário, não use esta flag), para a origin (nome/endereço do repos. github que criamos), o master (foi editado por último, o branch que você está), em seguida será solicitado usuário e senha do Github, caso não tenha usado -u. Pronto ! OBS: Caso efetue mais alterações, execute este comando novamente sem o -u, para enviar as novas alterações.

EXECUTAR UM FORK EM UM REPOSITÓRIO

Esta opção, realiza uma cópia de um projeto que não é seu, para a sua máquina, e depois das alterações efetuadas você pode subir e efetuar um **Pull Request** no repositório, este se diferencia do git clone, com a diferença de que no clone o repositório é seu, logo será possível atualizar o repositório, e no Fork, necessita executar o *pull request*, para que o proprietário do repositório avalie, e se assim desejar implemente ao projeto suas modificações.

SIGNIFICADO DE BRANCH

Branch (ramo) é simplesmente um leve ponteiro móvel para um dos commits. O nome do branch padrão é **Master**. Como você inicialmente fez commits, você tem uma branch principal (**Master Branch**) que aponta para o último commit que foi realizado, este commit é identificado através da hash gerada ao commitar, e pra cada hash ele tira uma foto(SNAPSHOT) de tudo que está naquele estado. *Cada vez que se realiza um commit ele avança automaticamente.*

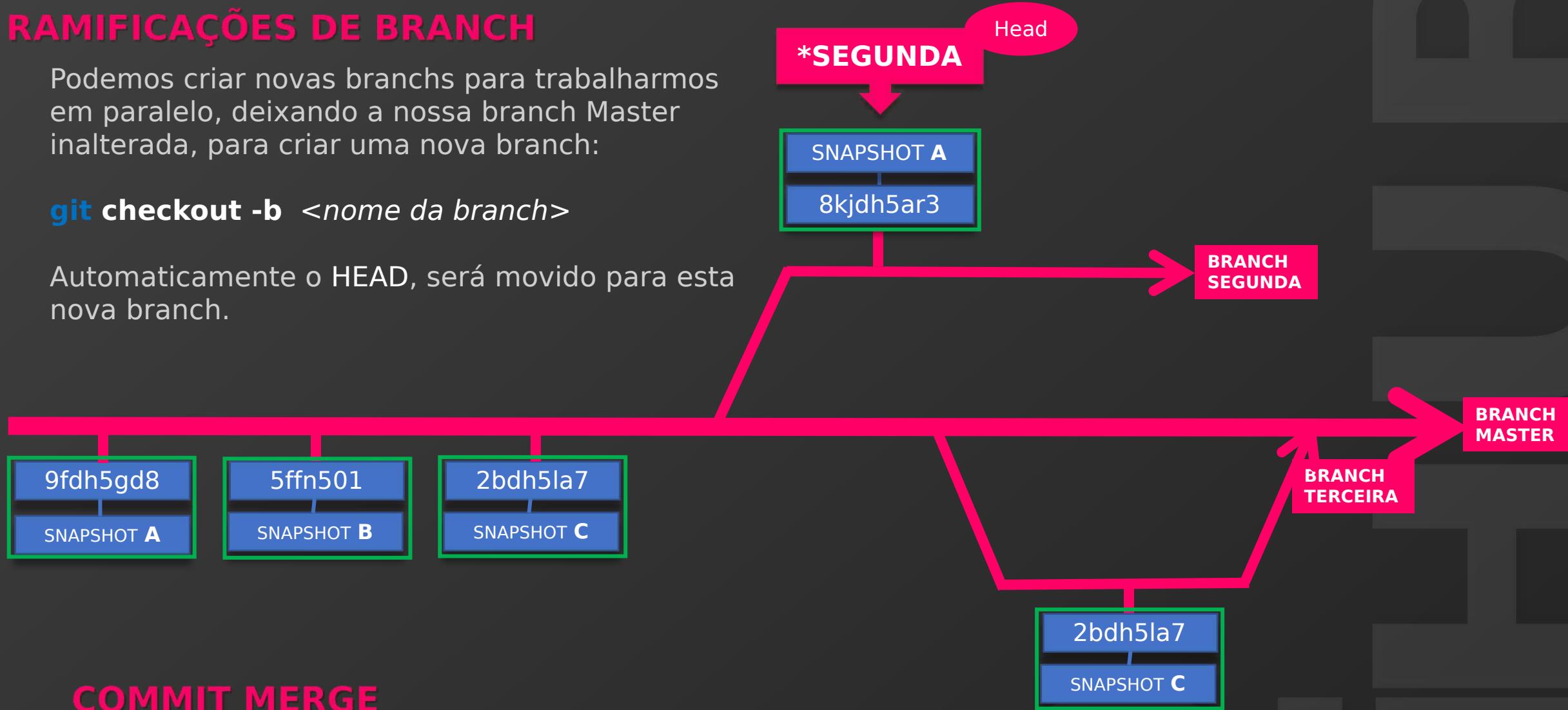


RAMIFICAÇÕES DE BRANCH

Podemos criar novas branches para trabalharmos em paralelo, deixando a nossa branch Master inalterada, para criar uma nova branch:

git checkout -b <nome da branch>

Automaticamente o HEAD, será movido para esta nova branch.



COMMIT MERGE

É um commit realizado para unir a sua branch criada em paralelo com a branch Master:

git merge <nome da branch a unir com Master>

INFORMAÇÕES

ADICIONAIS



FORMAS DE ESCRITA

CASE SENSITIVE é a prática de fazer distinção sobre palavras escritas ou iniciadas com maiúsculas ou minúsculas, ou seja a palavra 'sum' é diferente de "SUM".

Case Sensitive: “CARRO” != “carro”

CAMEL CASE é a prática de escrever palavras compostas ou frases de modo que cada palavra ou abreviatura no meio da frase comece com uma letra maiúscula.

Camel Case: “iPhone”, “sobreNome”

PASCAL CASE é a prática de escrever palavras compostas ou frases de modo que cada palavra ou abreviatura comece com uma letra maiúscula.

Pascal Case: “FedEx”, “HarperCollins”

CAPITALIZE é a prática de escrever a primeira letra de uma palavra com a letra maiúscula.

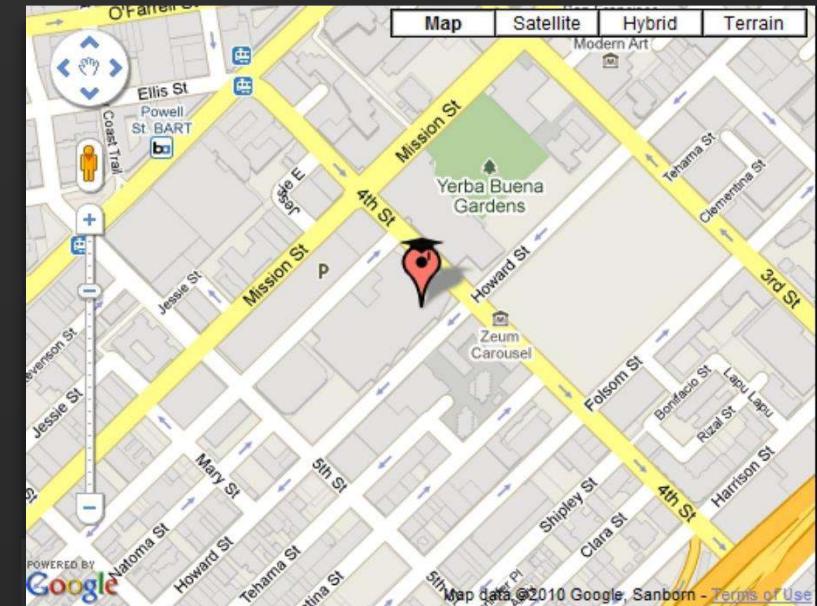
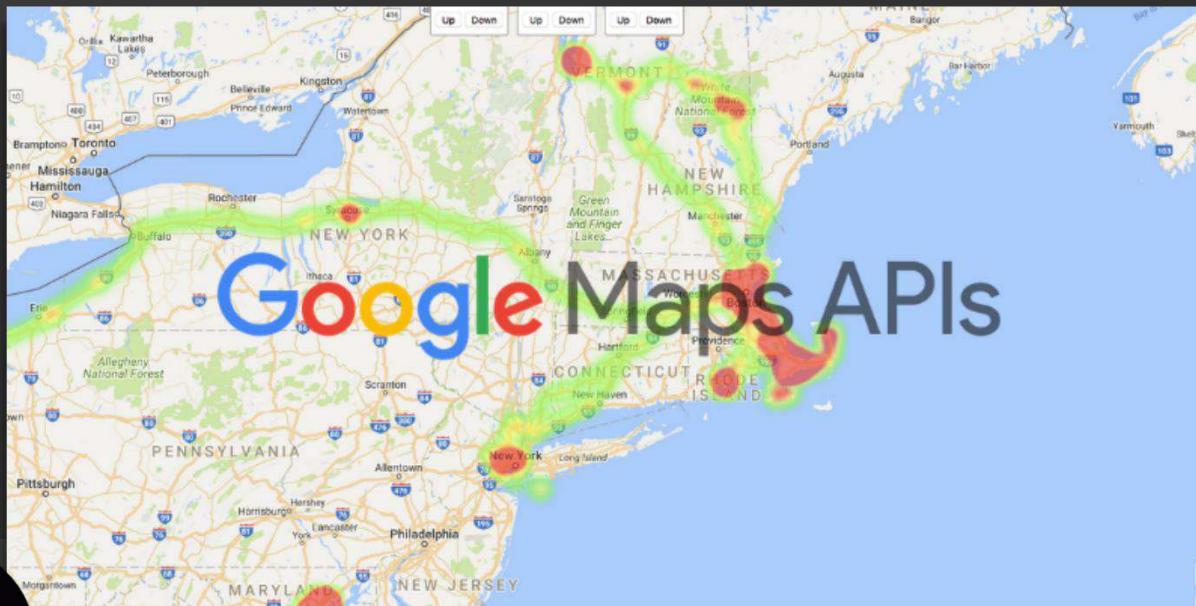
Capitalize: “Carro”, “Rafael”

UPPERCASE e **LOWERCASE** é prática de escrever as palavras com letras maiúsculas ou minúsculas respectivamente.

Uppercase: “CARRO”, “SOFÁ” Lowercase: “cama”, “mesa”

API

Uma API é criada, quando uma empresa tem a intenção de que outros criadores de software desenvolvam produtos associados ao seu serviço. Existem vários deles que disponibilizam seus códigos e instruções para serem usados em outros sites. O Google Maps é um exemplo na área de APIs. Por meio de seu código original, muitos outros sites e aplicações utilizam os dados do Google Maps adaptando-o da melhor forma a fim de utilizar esse serviço, como por exemplo usá-lo no site para mostrar onde se localizam, o desenvolvedor apenas utiliza a API do Google, sem que precise criar o código fonte.



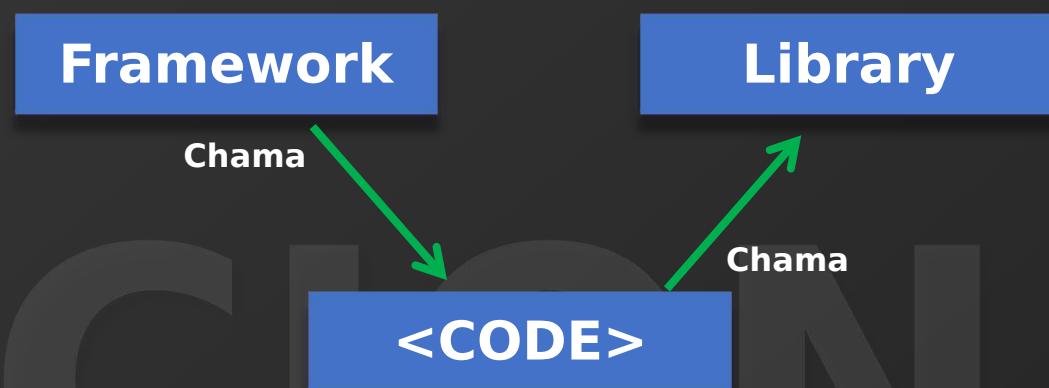
FRAMEWORK / BIBLIOTECA

A principal diferença entre uma Biblioteca e um Framework é "Inversão de controle". Quando você chama um método de uma biblioteca, você está no controle. Mas com um framework, o controle é invertido: a estrutura chama você.

Uma **biblioteca** é apenas uma coleção de definições de classe, a razão por trás é simplesmente a reutilização de código. Por exemplo, existem algumas bibliotecas de matemática que podem permitir que o desenvolvedor chame a função sem refazer a implementação de como um algoritmo funciona.

Em um **Framework**, todo o fluxo de controle já existe e há vários espaços em branco predefinidos que você deve preencher com seu código, ele define um esqueleto em que o aplicativo define seus próprios recursos para preencher o esqueleto.

Pode-se dizer que um Framework é um conjunto de bibliotecas.



FRAMEWORK CSS



Bootstrap 4



Materialize



Foundation
ZURB

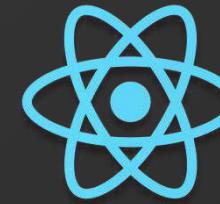


PLATAFORMA
DEV

FRAMEWORK JS



Vue.js



React



Angular



LIBRARY JS



chart.js



three.js

jQuery



SPA

SPA ou Single Page Application (Aplicação de Página única), todo o código necessário HTML , JavaScript e CSS, é recuperado com um único carregamento de página, qualquer interação que aconteça na página, serão carregadas dinamicamente, e adicionadas. A página não é recarregada em nenhum momento do processo, nem controla a transferência para outra página, a interação, geralmente envolve comunicação dinâmica com o servidor da web nos bastidores.

Em resumo possuímos apenas uma página index.html, e todo o resto da aplicação é gerada a partir do javascript, e o acesso ao BD é dinâmico e quase instantâneo



source: <https://dotcms.com/blog/post/what-is-a-single-page-application-and-should-you-use-one>

UML

Unified Modeling Language(Linguagem de Modelagem Unificada), basicamente é uma linguagem de notação (um jeito de escrever, ilustrar, comunicar) para uso em projetos de sistemas, como o nosso sistema será desenvolvido e todas suas funções.

Porque usar?

- Solução de Problemas, reduz os custos dos produtos nas empresas e aumenta a qualidade
- Melhorar a produtividade, ao usá-lo todos na equipe estão na mesma página.
- Fácil de entender, oferece uma apresentação clara e expressiva, dos requisitos, processo e funções do sistema.

Principais Usos

- Rascunho do sistema, o diagrama é usado pela equipe para estruturar o sistema geral
- Visualizar linguagem de programação, alguns tipos de diagramas podem ser traduzidos diretamente em código para economizar tempo no desenvolvimento do software

Tipos de Diagramas

As duas principais categorias são de Estrutura e de Comportamento, sendo que estas possuem mais 13 subtipos.

- Estrutura: Diagrama de Classe, Diagrama de Estrutura Composta, Diagrama de Objeto, Diagrama de Componente, Diagrama de Implementação, Diagrama do Pacote,
- Comportamento: Diagrama de Máquina de Estado, Diagrama de Atividades, Diagramas de Casos de Uso, Diagrama de Sequência, Diagrama de Comunicação, Diagrama de tempo, Diagrama de Visão Geral da Intereração.

PARADIGMAS DE LINGUAGEM

P
A
R
A
D
I
G
M
A
S

IMPERATIVO

ESTRUTURADO

DECLARATIVO

ORIENTADO A OBJETOS

FUNCIONAL

LÓGICO

Dividido em Blocos

Sequência, Seleção e Iteração

Abstração de Dados

Encapsulamento, Comportamento
de dados e objetos

Envio de mensagens entre
objetos

As funções são elementos de
Primeira Ordem

Uso de cálculo Lambda

Os programs são construídos
de predicados e relações

NODE.JS

Node Js é uma plataforma em tempo de execução, que usa javascript no servidor. Foi introduzida pela primeira vez em 2009, por Ryan Dahl.

Quanto as **vantagens** do Node:

- O profissional mantém-se em apenas em uma stack, pois utiliza javascript tanto no backend quanto no frontend
- Compartilhamento e reutilização de código
- Velocidade e desempenho
- Entrada e Saída sem bloqueio e a manipulação de solicitações assíncronas
- Tecnologia escalável para o uso de microserviços
- Desenvolvimento de aplicações em tempo real, melhor desempenho para jogos online.

Quanto as **desvantagens**:

- Não é recomendado para computação pesada, por incapacitação de tarefas ligadas à CPU
- Por ser assíncrono, depende muito de retorno de chamadas (callbacks), podendo resultar em uma Callback Hell, se o programador não se atentar a este ponto
- Valor elevado para seu uso em servidores

DESIGN PATTERN & ARCHITETURAL PATTERN

Architetural Pattern é semelhante ao Design Pattern, mas eles possuem uma diferença de escopo, o **Design Pattern** afeta uma seção específica da base de código, pois é uma solução de projeto que se aplica à um problema comum, que se repete em um projeto de software, o **Architetural Pattern** é uma estratégia de alto nível que diz respeito a componentes de larga escala, às propriedades e mecanismos globais de um sistema, é um padrão de organização estrutural, a arquitetura usada no desenvolvimento, podendo ter mais de uma.

DESIGN PATTERN

Categorias:

Criacionais: Singleton, Builder, Prototype, etc

Estruturais: Composite, Facade, Proxy, etc

Comportamentais: Mediator, Observer, Strategy, etc

ARCHITETURAL PATTERN

Layered, Event-Driven, Microservices, MVC, etc

MVC

MVC significa Model View Controller, É **Architecture pattern**, que nos permite ter uma organização adequada do nosso código, separando as responsabilidades.

