


```
</colgroup>

<thead>
  <tr>
    <th>Monday</th>
    <th>Tuesday</th>
    <th>Wednesday</th>
    <th>Thursday</th>
  </tr>
</thead>
<tbody>
  <tr>
    <td>...</td>
    <td>...</td>
  </tr>
  <tr>0
</tbody>
<tfoot>
  <tr>
    <td>...</td>
    <td>...</td>
  </tr>
</tfoot>
</table>
```

```
• <video src="path to video here" controls> ps: controls used so that browser offers
  controls for video
  <p>this text will appear if your browser doesn't support the video element</p>
</video>
-> To prevent issue of video not playing because of the format
<video controls>
  <source src="URL" type="video/mp4"/>
  <source src="URL" type="..." />
  <p>text displayed due to video not playing due to format</p>
</video>
-> Other video features
<video controls width="400" height="400" autoplay loop muted preload="_3choices_"
poster="URL">
  <source ... /> ... ... </video>
<!--width and height in CSS pixels, autoplay: starts playing right away while rest of
page is loading, muted as default in the beginning, preload: none means not
preloaded, metadata means gets only metadata preloaded, auto means whole video
downloaded is preloaded, if empty means it's auto-->
```

```
• <audio controls>
  <source src="URL" type="audio/mp3"/> ...
</audio>
<!--Add text tracks to video --->
<video controls>
  <source src="_" type="_"/>
  <track kind="subtitles, caption or description" src="name.vtt"
  srlang="port-br" label="language name here"/>
</video>
```

Embed PDF...

- <iframe> allows to embed other web documents into the current document(incorporate 3° party content)
<iframe scr="_" width="100%" height="1=500" allowfullscreen sandbox>

<p> fallback link for browser that don't support iframes </p>
</iframes>
ps: to improve speed set iframe's scr attribute with JavaScript after main content is alread loaded
ps: iframes are a common attack vector so they maliciously modify your page or tuck people into doing something they don't want to
-> HTTPS is the encrypted version of HTTP, so use it to serve your website
-> reduces chance that remote content has been tampered with during transit
-> prevents embedded content from accessing content in your parent document and vice versa
-> HTTPS enabling your site requires special security certificate to be installed
Many hosting providers offer HTTPS support for your site (like github), if you want to set up HTTPS support on your own you can use Let's Encrypt tools and instructions
<object data="mypdf.pdf" type="application/pdf" width="800" height="1200">
<p>plugin</p> </object>
<embed scr="mypdf.pdf" type="_" width="_" height="_"/>
Link relations
2 types: **to external resources and hyper links**

CSS

Descreve a aparência e a disposição das informações
Pode ser usado de 2 formas: incorporado ao HTML ou em um arquivo .css
Vantagens de separar conteúdo de estilo: melhor reuso, regras de estilo separadas podem ser usadas para diferentes tipos de media, diminui consumo de largura de banda pois folhas de estilo são armazenadas em cache

- Introdução

Sintaxe básica (3 partes)

p { color: green; }
Seletor (**elemento HTML** identificado por sua **tag, classe ou ID**, a regra será válida a ele)

Propriedades (atributo do elem HTML ao qual será aplicado a regra)
Valor (característica específica a ser assumida pela propriedade)

Tipos de folha de estilo (onde as regras estão descritas)
Folha de estilo **externa**: arq externo referenciado através da <link> ou utilizando a regra @import

- ideal para ser aplicada a várias páginas
- elem **link** dentro do elem **head**
- elem **style** dentro elem **head**

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="/style.css" media="screen" >
  </head>
</html>

atributo rel é obrigatório, adiciona metadados (css sempre "stylesheet")
```

<!DOCTYPE html> <html> <head> <style media="screen, projection"> @import url("estilos.css"); </style> </head> </html>	atributo media especifica a que tipos de media o css será aplicado, caso omitido será all opções: all, print(pré-visualização de impressão), screen(em dispositivos com tela), speech(leitores de tela) ps: between 2 media types operator allowed are and not and ,(=or)
--	--

Folha de estilo **interna**: separadas do conteúdo em HTML, mas contidas no próprio doc web utilizando a tag <style>

- ideal para ser aplicada a uma única página
- declaradas na seção **head** (tb necessita de atrib **media**)

```
<!DOCTYPE html>
<html>
  <head>
    <style media="all">
      body {
        background-color:#000 ;
        color: red;
      }
    </style>
  </head>
</html>
```

Folha de estilo em **linha**(inline): usando atributo style nas tags HTML

- se aplica a um elem HTML
- ã pode usar atributo media, ã é aplicável a pseudo-classes e pseudo-elementos

```
<!DOCTYPE html>
<html>
<body>
  <p style="color: blue; background-color: yellow;">
    O conteúdo desta tag será exibido em azul com fundo amarelo no navegador!
  </p>
</body>
</html>
```

Folha de estilo externa

Ideal para ser aplicadas a várias pags, deve ser linkado ou importado ao doc HTML (link deve estar dentro do elem head; style tb deve estar contido no head do doc)
<link rel="stylesheet" href="/style.css" media="screen">
ps: rel É obrigatório, sempre "stylesheet"

Seletores

Aplica a regra a todos elem citados que estão no arq HTML
NÃO são case sensitive (boa prática usar letras minúsculas)
p {color: #5EC85E;}

#id_do_elemento{color: #5EC85E;}

Por id:

p {color: purple;} #missao {color: orange;}	<p> Cidade de Esparta </p> <p id="missao" > A nossa missão é
--	---

	fornecer mais espartanos </p>
resultado:	Cidade de Esparta <p>A nossa missão é fornecer mais espartanos</p>

Por classe(como base atrib classe do elem):

Aplicável em elem com + de 1 classe

p.red {color: #f33;} .yellow-bg {background-color : #ffa;} .fancy {font-weight: bold;text-shadow: 4px 4px 3px blue;}	<p class="red">This paragraph has red text.</p> <p class="red yellow-bg">This paragraph has red text and a yellow background.</p> <p class="red fancy">This paragraph has red text and "fancy" styling.</p>
.warning {background-color: red;} p.warning {color: white;} .special {text-shadow: 4px 4px 3px blue;font-weight: bold;} .warning.special {color: blue;}	<p class="warning"> Texto 1</p> <h1 class="warning"> Texto 2</h1> <p class="warning special"> Texto 3</p>

Por atributo:

Combinando elem + presença de um atributo ou de acordo com o valor do atrib
tag-name[atributo= “valor”] {text-align: center;}
ps: atributo pode ter qualque nome, no exemplo href
[attribute^="specifiedvalue"]: selects elem whose attrib value **starts with the specified value**
[attribute*="value"]: selects elem whose attrib value **contains specified value**
[attribute\$="value"]: selects elem whose attrib value **ends with specified value**
[attribute="specifiedvalue"]: selects elem whose attrib value **has exact same specified value**
[attribute~="specifiedvalue"]: selects elem whose attrib value **space separates exact same specified value**
[attribute]: selects elem wiith this attrib

CSS a[href^="#"] {background-color: gold;} a[href="example"] {background-color: silver;} a[href*="insensitive" i] {color: cyan;} a[href\$=".org"] {color: red;} a[href^="https"][href\$=".org"] {color: green;} /* not case sensitive*/ HTML Interno Exemplo Insensitive internal Exemplo org Exemplo https org 	Interno Exemplo insensitive Interno Exemplo org Exemplo https org
--	--

Por Universal

Irá escolher todos os elementos HTML

*** { text-align: center; }**

***[lang^="en"] { color: green; }**

Agrupando seletores: grupo, descendência, filhos, irmãos, irmãos adjacentes

Grupo

elemento1, elemento2 { color: green; }

ex: **h1, h2, span .titulo { color: red; } //titulo é a classe**

Descendência

Alvo bem especificado, pode descrever elem contido em outro elem

• É a combinação de **2 ou + seletores** simples **separados** por um espaço em branco

• Combina elementos que sejam **descentes** do primeiro elemento

elemento1 elemento2 { color: green; }

ex: **#ad li.important strong { color: green; }**

<div id="ad"> <p> MercadinhoCompre aqui </p> <li class="important"> Os melhores preços da cidade! Enquanto durar o estoque! </div>	Mercadinho Compre aqui Os melhores preços da cidade Enquanto durar o estoque
---	---

Filhos

Alvo: Só os filhos de um elem

um ou + seletores separados por >

elemt_father > elem_child { color: green; }

Elements that are not directly a child of the specified parent, are not selected.

ex: **strong{ color: green; }**

div > strong { color: purple; }

<div> Eu gosto de uva roxa <p>Eu prefiro uvaverde</p> </div>	Eu gosto de uva roxa Eu prefiro uva verde
--	---

Irmãos

Alvos: irmãos do elem especificado anteriormente e que aparecem depois dele separado por ~

elemento1 ~ elemento2 { color: green; } //logo, todos elem2 depois da existencia anterior de ao menos um elem1

span{ color: blue; } p ~ span { color: red; }	dfaefaaavsdvdsvf//blue <p>The first paragraph.</p> dfaefaaavsdvdsvf//red dfaefaaavsdvdsvf//red
---	---

Irmãos Adjacentes

Alvo: um elem que seja irmão e adjacente ao primeiro

span{ color: blue; } p + span { color: red; }	Irmão que vem antes.//blue <p>Irmão principal.</p> Irmão que vem depois!//red Irmão//blue <code>Irmão que vem depois!</code> Outro irmão//blue <p>Outro irmão principal.</p> Último irmão.//red
---	--

important :first-of-type

table:not(:first-child) { font-weight: bold; }

- Pseudo-classes**

Similares a uma classe CSS, porém não são especificadas na marcação

Seleciona elementos HTML baseados em seu estados (ex:se link foi visitado)

seletor com : seguido do nome da pseudo-classe

seletor:pseudoclesse {... }

podem ser dinâmicas

aplicadas de acordo com a interação do usuário com o documento

podem classificá-las em 9 categorias:

- Estados de visualização (Element display state),
- Entrada de dados (input),
- Linguística,
- Localização,
- Estado de Recursos (resource state),
- Dimensão temporal (time-dimensional),
- Estruturais (tree-structural),

8. Ações do usuário

9. Funcionais.

Exemplo de alguns:

:disabled	Entrada de dados	É aplicada em elementos de entrada desabilitados
:invalid	Entrada de dados	É aplicada em elementos de entrada que inválidos de acordo com validação
:lang()	Linguística	Seleciona um elemento de acordo com o seu idioma
:link	Localização	É aplicada em links não visitados
:visited	Localização	É aplicada em links já visitados
:playing	Estado de recursos	É aplicado em um elemento de média quando seu conteúdo está em reprodução
:empty	Estrutural	É aplicado em elementos que não possuem filhos
:active	Ações de usuário	Link no momento em que ele é clicado
:hover	Ações de usuário	Quando usuário põe o cursor em cima de um elemento
:focus	Ações de usuário	Quando um elemento recebe foco

p:first-child { color: red; } p:nth-child(2n) { color: green; } //2nd, 4th, 6th, 8th, 10th... children p:nth-child(3) { color: orange; } p:last-child { text-align: center; }	<p>This text is RED!</p> <div> <p>This text is RED!</p> <p>This text is GREEN</p> <h2>This text isn't selected: it's not a `p`.</h2> <p>This text is GREEN</p> <p>This text is black</p> <p>This text is GREEN</p> <p>This text is black in the center</p> </div>
--	---

.button {border-radius: 5px;border: 1px solid gray;padding: 5px; a.button:link {color: gray;text-decoration:none; a.button:visited { color: black;} a:hover { background-color: gray;}	Ação 1 Ação 2 Ação 3
--	---

- Pseudo-elementos**

Used to style specified parts of an element.

Used to: style 1º letter, line or elem; insert content before or after content of an element

p::first-line { color: red; } //formats 1º line of text in <p> elems

Exemplo de alguns:

::after	Insere alguma coisa depois do conteúdo do elemento selecionado
::before	Insere alguma coisa antes do conteúdo do elemento selecionado
::first-letter	Aplica a regra na primeira letra do conteúdo de texto de elemento selecionado

::first-line	Aplica a regra na primeira linha do conteúdo de texto de elemento selecionado
::marker	Aplica a regra nos marcadores dos itens de listas
::selection	Aplica a regra na porção do conteúdo selecionado de um elemento

a::after { content: “(”attr(href)””); }

• O efeito em cascata

Estabelecimento de uma prioridade para aplicação de regras de estilos ao elementos

CSS in HTML File WINS external CSS file-> always

Se ainda precisar...Calcule Especificidade da declaração css: considera 4 colunas

Inline - ID(seletores e ID) - Classe(classe, atrib, pseudo-classes) - Tipo(seletor de elem e pseudo-elem)

.special.box { color: red; }//0 -0 - 2 - 0 #b { color: orange; }//0 - 1 - 0 - 0 //pink wins (biggest num wins))	<div class="box special" id="b" style="color:pink;">Caixa 4</div>
--	---

ps: !important rule overrides inline styles; inline = 1000,id=100,class=10,type=1

ps: *universal selector is ignored; coluna +baixo **NUNCA** vence +alta

caso de **empate última declaração vence**

span { color: blue; border: 1px solid black; } .extra span { color: inherit; }	<div style="color:red"> Here is a span which is blue, as span elements are set to be. </div> <div class="extra" style="color:green"> Here is a span which is green, because it inherits from its parent. </div>
---	--

[filter \(efeitos especiais\)](#)

• CSS Box Model (wraps around every HTML element)

Outside to-inside (4):

Margin(clears area outside border. always transparent),

margin-top ; margin-right; margin-bottom; margin-left OR
margin: auto//horizontally center the element within its container, elem will take up
this width, the remaining space will be split equally between the left and right
margins.

margin: inherit;

margin: 1em//1 valor aplica em todos os lados

margin: 25px 50px//2 valores: vertical horizontal

margin: 20px 30px 20px //3 valores: topo horizontal inferior

margin: 10px 15px 20px 30px//4 valores: topo direita inferior esquerda

margin-{top, right, left, bottom}: ...

Border(boxes the around of padding, padrão: transparentes e espessura zero),

border: solid; //1 value: style

border: outset #f33; //2 value: style, color OR width, style

border: 2px dotted;

border: medium dashed green; //3 value: width, style, color

border: 2px solid red; //4 values: width, style, color, color

border-width: r; //medium is default

border-style: r;//none is default

border-color: red;//currentcolor is default

border-{top, bottom, right, left}-{width, style, color}: {medium, none, currentcolor}

Padding(area around content, padrão: transparentes e espessura zero),

padding: 1em;//1 value Apply to all 4 sides

padding: 5% 10%; //2 values: 1º is top and bottom, 2º is left and right

padding: 1em 2em 2em;//3 values: 1º is top, 2º is left and right, 3º is bottom

padding: 5px 1em 0 2em;//4 values: top, right, bottom, left

//Properties, you can also set padding area with proprieties insted

padding-{top, bottom, right, left};

Content(text, img...)

If **box-sizing: content-box;//default**

width: 5px;

min-width: 5px;

max-width: 5px;

height: 5px;

min-height: 5px;

max-height: 5px;

ps: Cada elemento possui uma margem determinada por padrão

padding: 1em //apply to aplicativo 4 sides

.box {

box-sizing: content-box OR border-box;

//content box() width and height are just the content W/H

//borde box width and height margin(all the elements W or H will sum to that value)

background-color: lightgrey; //se refere a padding+content

width: 300px;

height: 150px;

padding: 50px;

border: 15px solid green;

margin: 20px;

}

Outline(line drawn outside the element's border)

outline-style; outline-color; outline-width;

outline-offset(specifies space between an outline and the edge
or border of an element); outline(shorthand for outline-width,
outline-style, and outline-color)

• Elementos flutuantes

Float permitem colocar elem de bloco lado a lado

float: none|left|right|initial|inherit;

margin: 0 auto;

only works if there's width.

as vezes fica em cima/'come' de outro elemento, as vezes ã

Clear desabilita a sobreposição de um elemento flutuante sobre esse elemento

clear: left|right|both|none|inherit //none is default

none - element is not pushed below left or right floated elements.

left - element is pushed below left floated elements

right - element is pushed below right floated elements

both - element is pushed below both left and right floated elements

Overflow é utilizado quando elem flutuante é maior que a área ocupada pelo conteúdo do seu elem pai. Especifica o que deve ocorrer com o elem filho muito grande

overflow: hidden|visible|scroll|auto //visible is defalut, auto similar to scroll but only

when necessary

visible: is not clipped, the content renders outside the element's box

hidden: overflow is clipped, rest of content invisible

scroll: is clipped, ans a scrollbar is added

auto: similar to scroll, but it adds scrollbars only when necessary

ps:overflow property only works for block elements with a specified height.

overflow-x: specifies what to do with the left/right edges of the content.

overflow-y: specifies what to do with the top/bottom edges of the content.

choices for both overflows: **hidden|visible|scroll|auto**

Grid of boxes/equal width boxes

*** {box-sizing: border-box;}**

//default is content-box where width/height of elem = width/height + padding +
borderbox-sizing allows us to include the padding and border in the box's total width/
height, making sure that the padding stays inside of the box and that it does not
break

However, this is not very flexible. If boxes have different content sometimes the one
with more will have his content displayed outside of the box. This is where Flexbox
comes in handy - as it can automatically stretch boxes to be as long as the longest
box:

Display property: specifies how an element is shown on a web page

(controlling layout). specifies the type of box used for an HTML elemnt

has many values, some: inline, block, contents, flex(displays an element as a
block-level flex container), grid (displays an element as a block-level grid
container)...

display: none is commonly used in JavaScript to hide and show elements, doesn't
take up space while visibility: hidden; does

• Flexbox:

Utiliza 2 tipos de “caixas”: containers e itens

Containers: mostram itens e posiciona-os. Pai com display flex(**display: flex;**)

Item é elem html filho direto de um contêiner (so automatically flex children).

Possível mesclar contêineres flex e contêineres regulares

Containers Properties:

- flex-direction**: which direction the container wants to stack the flex items.
flex-direction: column|column-reverse|row|row-reverse;
- flex-wrap** whether flex items should wrap or not, if there is not enough room for them
on one flex line
flex-wrap: wrap|nowrap(default)|wrap-reverse;
nowrap: flex items will not wrap no matter what //DEFAULT
wrap-reverse: wrap if necessary, in reverse order
- flex-flow** shorthand, sets **flex-direction** and **flex-wrap** properties
flex-flow: column|column-reverse|row|row-reverse wrap|nowrap(default)|wrap-reverse;
- justify-content**: Horizontally aligns the flex items when the items do not use all
available space on the main-axis
flex-contaimer: center|flex-start|flex-end|space-around|space-between;
- align-items** Vertically aligns the flex items when the items do not use all available
space on the cross-axis
align-itens: center|flex-start|flex-end|strectch|baseline;
- align-content** Modifies the behavior of the flex-wrap property. It is similar to
align-items, but instead of aligning flex items, it aligns flex lines

align-content: space-between|space-around|fstrectchcenter|baseline;flex-start|flex-end

Itens Property:

- order: order value must be a number, default value is 0 **order: 1;**
- flex-grow: specifies how much a flex item will grow relative to the rest of the
flex items. Default is 0.
flex-grow: 1;

3.

flex-shrink: specifies how much a flex item will shrink relative to the rest of the flex items. Default is 1.
4.

flex-basis: initial length of a flex item (any unit od mesurement)
5.

flex: shorthand for flex-grow, flex-shrink, flex-basis
6.

align-self: alignment of the item inside the flexible container.
- align-self: centerflex-startflex-end;

FLEX RESPONSIVE

Position property specifies the type of positioning method used for an element

- 5 types:
1.

position: static;(default): it is always positioned according to the normal flow of the page. Not affected by top, bottom, left, right...
2.

position: relative: positioned relative to its normal position (using left,right, top, bottom) Other content will not be adjusted to fit into any gap left by the element.

- ex:left: 40px;
3.

position: fixed: relative to the viewport, which means it always stays in the same place even if the page is scrolled
4.

position: absolute: relative to the nearest positioned ancestor(unless it's static), if there are no ancestors it uses the body and moves along with the page scrolling

ps: those elem are removed from normal flow, so they can overlap elements

5.

position: stickyBased on user's scroll position. In the beginning it's relative after scrolling the element it becomes fixed
- Z-index (elem que usam relative, absolute and fixed tem)

- Responsividade
- Media query: permite aplicar regras mediante uma condição
- @media <media_type> and <media_feature> {
- <Regras_CSS>}

Exemplo:

```
@media only screen and (max-width: 400px) {
body { background-color: #F09A9D; } }
@media only screen and (min-width: 401px) and (max-width: 960px)
{ body { background-color: #F5CF8E; } }
@media only screen and (min-width: 961px)
{ body { background-color: #B2D6FF; } }
```

Tipos de Layout: fluido, largura-fixa

Mobile-first pois é menos complexo (usa min-width) laptop usa max-width

GRID	FLEXBOX
<div>* Grid is made for a two-dimensional layout while Flexbox is for one. This means Flexbox can work on either row or columns at a time, but Grids can work on both</div> <div>* Allows you to place elements on different sections of the browser page while maintaining proper alignment</div> <div>* better responsive (more room for flexibility and resizing content)</div> <div>* Flex leaves some white space at the extreme, the CSS grid controls white space by distributing elements equally along the row and also based on the allocated column space.</div>	<div>popular for its order ability (enables the developer to reorder content without having to change the HTML content manually)</div>
<div>ex:</div> <div><pre>.main{ display: grid; grid: auto auto/auto auto auto auto; grid-gap: 10px;</pre></div>	<div>ex:</div> <div><pre>.main{ display: flex; grid: auto auto/auto auto auto auto; grid-gap: 10px;</pre></div>

<pre>background-color: green; padding: 10px; } .gfg { background-color: rgb(255, 255, 255); text-align: center; padding: 25px 0; font-size: 30px; }</pre>	<pre>background-color: green; padding: 10px; } .gfg { background-color: rgb(255, 255, 255); text-align: center; padding: 25px 0; font-size: 30px; }</pre>
---	---

- Modelos de conteúdo:
- elementos do HTML podem ser agrupados em 15 categorias que compartilham características semelhantes, cada elemento pertence a zero ou mais categorias

Layouts com HTML semântico (transmitem o significado): Tags semânticas para texto, tags semânticas para estrutura, Section elements

<article> (An article should make sense on its own, and it should be possible to distribute it independently from the rest of the web site. used for: blog posts, user comments, product cards, newspaper articles...),**<address>**, **<aside>**, **<details>**, **<figcaption>**, **<figure>**, **<footer>**, **<header>**, **<main>**, **<mark>**, **<nav>**, **<section>** (defines a section in a document, that doesn't need to make sense without context of doc; thematic grouping of content, typically with a heading; used for: chapters, introduction, news items, contact information), **<summary>**, **<time>**

- ps: Evite a manipulação do outline usando <section> pq é muito genérico. <section> em vez de <div>
- > Fornece informações adicionais que ajudam a definir as funções e a importância relativa das diferentes partes da sua página
- > Faz página mais acessível, é uma boa prática

Esboço do layout semântico:

É como os mecanismos de pesquisa e os leitores de tela visualizam a hierarquia do conteúdo da página

Todos os elementos de cabeçalho contribuem para o esboço do documento de uma página.

<div><h1>Meu HTML Semântico</h1></div> <div><p>Por Bruno Góis Mateus. Publicado em 2023</p></div> <div><p>Um belo exemplo de HTML semântico</p></div> <div><h2>O "esboço" do documento</h2></div> <div><p>O HTML5 inclui diversos elmentos (sectioning content) que afetam o outline do documento.</p></div> <div><h3>Headers</h3></div> <div><p>O <code>&lt;header&gt;</code> é um desses elementos.</p></div> <div><h3>Footers</h3></div> <div><p>O <code>&lt;footer&gt;</code> também.</p></div> <div><h2>Elementos semânticos inline</h2></div>	<div>Meu HTML Semântico</div> <div>O "esboço" do documento</div> <div>Headers</div> <div>Footers</div> <div>Elementos semânticos inline</div>
---	---

<div><p>O <code>&lt;time&gt;</code> também é semântico, mas não é um sectioning content.</p></div> <div><p>&copy; 2023 Bruno Góis Mateus</p></div>	
<div><article></div> <div><header></div> <div><h1>Meu HTML Semântico</h1></div> <div><p></div> <div>Por Bruno Góis Mateus. Publicado em</div> <div><time datetime="2023-02-27">2023</time></div> <div></p></div> <div></header></div> <div><p>Um belo exemplo de HTML semântico</p></div> <div>...</div> <div><footer></div> <div><address></div> <div>Em caso de dúvidas, entrar em contato com <a</div> <div>href='mailto:brunomateus@exemplo.com'></div> <div>Bruno.</div> <div></address></div> <div></footer></div> <div></article></div> <div><footer></div> <div><p>&copy; 2023 Bruno Góis Mateus</p></div> <div></footer></div>	

N entendi!!!

Passos:

1. Escolher os contêineres do layout
2. Nomeá-los
3. Escrever HTML
4. Preencher com conteúdo mín
5. Posiciona-los

Cores

palavras-chaves: aqua, blue, navy, black, white, fuchsia, gray, lime, green, maroon, olive, orange, purple, red, teal, silver, yellow...

código rgb ou rgba: red green blue de 0 à 255

coordenada cilíndrica: hsl ou hsla

p { color: rgba(0, 0, 255, 1.0);

background-color: yellow;

border: 10px dashed #008000; }

Ferramentas Online: https://colorshemesigner.com/csd-3.5/ ; https://coolers.co

Fontes

p { font-family: Garamond, "Times New Roman", serif ; //fonte a ser usada

font-size: 2em; //tamanho das letras

font-weight: bold; //habilitar ou ão bold

font-style: italic; //habilitar ou não itálico

Propriedades de texto

p { text-align: left; //alinhamento

text-decoration: underline; //decorações

line-height: 1px; //espaçamento

word-spacing: -1px;

letter-spacing: -1px;

text-indent: 50px; } //margem antes da 1 letra do parágrafo

- CSS p/ Listas

[list-style](#) Sets all the properties for a list in one declaration

[list-style-image](#) Specifies an image as the list-item marker

[list-style-position](#) Specifies the position of the list-item markers (bullet points)

[list-style-type](#) Specifies the type of list-item marker

All border [properties](#).
Shadow

Boa práticas

- * Usar 1 ou + folhas de arquivo externa (mostra separação entre conteúdo e apresentação)
- * Folhas de estilos internas são justificadas se usadas em páginas que tenham uma aparência muito diferente do resto do site OU para serem combinandas com folhas externas
- * Folhas de estilo em linhas devem ser evitadas

CSS Reset devido cada navegador ter estilos diferentes definidos por padrão, o Reset seta um valor básico para todas as características do CSS, sobrescrevendo totalmente os estilos padrão do navegador.

links úteis: Borda, cores e esquema de cores • <https://html-css-js.com/css/generator/border-outline/> • <https://colorshemedesigner.com/csd-3.5/> • <https://coolors.co> • Fontes • <https://fonts.google.com> • CheatSheet • <https://htmlcheatsheet.com> • <https://htmlcheatsheet.com/css/> Efeito em cascata • <https://wattenberger.com/blog/css-cascade> • <https://specificity.keegan.st/> • Flex e Grid • <https://css-tricks.com/snippets/css/a-guide-to-flexbox/> • <https://css-tricks.com/snippets/css/complete-guide-grid/> • <https://www.origamid.com/projetos/css-grid-layout-guia-completo/>

Fundamentos de Bootstrap

Devido construir o layout e estilizar pag web com CSS demorar foi construído frameworks CSS, além disso tb facilitar a padronização

Framework CSS é um “arquivo css” que contém um série de regras CSS pré definidas

Alternativas: Tailwind CSS, Bulma, Foundation, Pure CSS, UI kit

Where to get Bootstrap:

Maneira + rápida de adicionar Bootstrap ao projeto: CDN (Content Delivery Network) -> jsDelivr (free open source CDN)

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-GLhITQ8iRABdZLI6O3oVMWSktQOp6b7In1ZI3/Jr59b6EGGol1aFkw7cmDA6j6gD" crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5MIQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CXvN" crossorigin="anonymous"></script>
```

Também é possível instalar via gerenciador de pacotes (npm, yarn ...) OU baixá-lo e usar diretamente no projeto

CSS files	Layout	Content	Components	Utilities
bootstrap.css bootstrap.rtl.css bootstrap.min.css bootstrap.rtl.min.css	Included	Included	Included	Included
bootstrap-grid.css bootstrap-grid.rtl.css bootstrap-grid.min.css bootstrap-grid.rtl.min.css	Only grid system	—	—	Only flex utilities
bootstrap-utilities.css bootstrap-utilities.rtl.css bootstrap-utilities.min.css bootstrap-utilities.rtl.min.css	—	—	—	Included
bootstrap-reboot.css bootstrap-reboot.rtl.css bootstrap-reboot.min.css bootstrap-reboot.rtl.min.css	—	Only Reboot	—	—
JS files	Popper			
bootstrap.bundle.js bootstrap.bundle.min.js	Included			
bootstrap.js bootstrap.min.js	—			

```
Template inicial
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"rel="stylesheet integrity="sha384-GLhITQ8iRABdZLI6O3oVMWSktQOp6b7In1ZI3/Jr59b6EGGol1aFkw7cmDA6j6gD" crossorigin="anonymous">
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5MIQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CXvN" crossorigin="anonymous"></script>
  </body>
</html>
```

Bootstrap Grid system (allows up to 12 columns)
ps: if you don't want to use 12 columns, then group them together (= wider column)
conceitos chaves: containers, grid, colunas e breakpoints

Breakpoints: blocos de construção para criar layouts responsivos quando aliados com media queries(min-width)

Breakpoint X-Small	Class infix None	Dimensions <576px
Small	sm ≥576px	
Medium md ≥768px		
Large lg ≥992px		
Extra large xl ≥1200px		
Extra extra large xxl ≥1400px		

Containers: bloco de construção que contém, preenchem e alinham seu conteúdo em um determinado dispositivo ou janela de visualização (/obrigatórios no Grid System

Existem 3 containers:
.container: configura o max-width de acordo com um breakpoint
.container-fluid: ocupa 100% da largura
.container-{breakpoint}: ocupa 100% da largura até o breakpoint especificado
{breakpoint} = sm || md || lg || xl || xxl || fluid

Colunas:

Basic Structure of bootstrap grip	
<pre><div class="container"> <div class="row"> <div class="col-*-*"></div> <div class="col-*-*"></div> </div> <div class="row"> <div class="col-*-*"></div> <div class="col-*-*"></div> <div class="col-*-*"></div> </div> <div class="row"> ... </div> </div></pre>	<p>1. create a container (<div class="container">).</p> <p>2. create a row (<div class="row">)</p> <p>3. add the desired number of columns (tags with appropriate .col-*-* classes).</p> <p>ps: numbers in .col-*-* should always add up to 12 for each row.</p> <p>ex: col-sm-3</p>

rebbot, tipografia, Listas

- .list-unstyled: Remove a margem e o marcado padrão
- .inline: Faz com que a lista ocupe uma linha
- Imagens
- .img-fluid: Torna a image responsiva.
- max-width: 100%; height: auto;

Tabelas

Helpers

- Clearfix (limpa elementos float)

```
<div class="clearfix">...</div>
```

```
@mixin clearfix() {
  &::after {
    display: block;
    clear: both;
    content: "";
  }
}
```

- Colored links(.link-*: para colorir links)

```
<a href="#" class="link-success">Success link</a>
```

```
<a href="#" class="link-danger">Danger link</a>
```

```
<a href="#" class="link-warning">Warning link</a>
```

```
<a href="#" class="link-info">Info link</a>
```

- Ratio
- Position
- Stacks
- Visually hidden
- Stretched link
- Text truncation
- Vertical rule

Utilities

JAVASCRIPT

- Linguagem de scripts, criada em meados dos 90s.
- Multiparadigma (funcional+imperativa).
- Tipagem dinâmica.
- Pode ser usado no lado do servidor “back-end”.
- Pode ser utilizado para desenvolvimento de aplicativos móveis.
- Permite a execução de scripts do lado do cliente
- Permitia a interação com o usuário sem a intervenção do servidor
- Lugar do código: parte inferior do HTML dentro da tag <script> </script>
- Accepts both double and single quotes

Principais features recentemente adicionadas: let/const, arrow functions, template literal, funções novas em arrays, default parameters, property shorthand

Tipos de dados: number, null, undefined, string, boolean, object, bigint, symbol

Number

ñ há separação entre números inteiros e números reais (são sempre prontos flutuantes), ambos são number. N° inteiros são precisos até 15 dígitos
Números especiais: NaN(not a number), Infinity
Operadores aritméticos: +, -, *, /, %, **, --. +=, -=, *=, /=, %=, **=

Mesma precedência do Java

Cuidado: Muitos operadores realização conversão automática de tipos

```
console.log(typeof 3.13); //number
console.log(typeof 5e-5); //number
L = 10.5; //console.log() writes to the console, used for testing F12 to see console view
```

BigInt

Números acrescidos de n no final
usados para representar números maiores de 2²³
operadores: +, -, *, /, **, console.log(typeof 1n) //bigint

Ao comparar com um number a conversão é necessária

Declaração de variáveis

3 modos de declarar: var (global), let (bloco), const (bloco, ñ pode alterar)

tipo de variável não é especificado

var msg = “oi”;

```
var n; //hoisting, can use variable before it's declared
```

```
var n = 4; //declaração
```

```
var n= “oi” ;//redeclaração
```

```
var n ; //even redeclaring 'var n' will not lose its value, CAN'T do this with let or cont
```

```
var person = "John", car = "Volvo", price = 200 ;//onse statement, many variables
```

ps: only use var if you MUST support old browser

JS treats _ and \$ as letter, so they are valid variable names

```
let n = 4; //ps: let CAN'T EVER be redeclared in same block
```

```
n = 5 ;//this you can: reassign
```

	Scope	Redeclare	Reassign	Hoisted	Binds this
var	global	✓	✓	✓	✓
let	✓	✗	✓	✗	✗
const	✓	✗	✗	✗	✗

Const

const must be assigned a value when declared const pi = 3.14;

can't reassign a const value/array/obj

doesn't define a const value, but const reference to a value

so you can change elem of const array, properties of const object

```
const cars = ["saab", "volvo", "bmw"];
```

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

```
cars.push= "audi";
```

Null is an object, no value

Undefined variáveis declaradas, ñ inicializadas ou membros arrays e objs que ñ existem

String principais métodos

length, concat(), indexOf(), lastIndexOf(), match(), replace(), slice(), split(), toLowerCase(), toUpperCase...

Can add strings ✓

```
let x= “oi” + “. ” + “tudo bem?” ;
```

```
let x= 5 + “2” + 3; //if one of the number in quotes all are treated as strings
```

Conversão String->Number

```
let count = 10;
```

```
let s1 = “” + count;//”10”
```

```
let s2 = count + “bananas”;//”10”
```

methods takes 1° number of string:

```
let s2 = parseFloat(“bananas");//NaN
```

```
let n1 = parseFloat(“36.11 43 12");//36.11
```

```
let n2 = parseInt(“12.2 32”, 10);//12, base of 10
```

Acessando caracteres

```
let letter = s[0]; //fails
```

```
let letter = s.charAt[0]; //works
```

If/else:

```
if (n == 1) {}
```

```
else if(n==2){}
```

```
else {}
```

Valores booleanos

valores considerados false = 0,0.0,NaN,””,null,undefined. Todo resto rue.

```
let b = Boolean(outroValor);
```

```
let n = true;
```

```
let n2 = “!E5” > 0;
```

```
if( “web dev”) // true
```

ps: == verifica a igualdade de valor e realiza a conversão de tipo, se necessário.

Operador === verificando a igualdade do valor e o tipo, sem realizar conversões automáticas.

ps: != tb ñ realiza conversão de tipos

```
”5.0” === 5//FALSE
```

Operadores lógicos: && || ! significa e, ou negação

Increment and Decrement:

let a = 3; //4 const b = ++a; //4	let a = 3;//4 const b = a++;//3
let x2 = 3n; const y2 = ++x2; // x2 is 4n; y2 is 4n	let a2 = 3n; const b2 = a2++; // s2 is 4n; b2 is 3n
let a = 3; const b = --a; //a:2, b:2	let x = 3; const y = x--; //x:2, y:3
let x2 = 3n; const y2 = --x2; // x2 is 2n; y2 is 2n	let x2 = 3n; const y2 = x2--; // x2 is 2n; y2 is 3n

operators

Switch/case

```
switch(n) {
  case 1: break; //faz esse se n === 1
  case 2: break; //faz esse se n === 2
  default: break; //se tudo falhar executa esse
}
```

Operador ternário, alternativa de if/else

```
const result = condition? trueExpression : falseExpression
```

```
const nota = 80
```

let conceito if (nota >70) {conceito=”bom”} else{conceito=”okay”}	const conceito = nota > 70? “bom” : ”okay”
---	---

Associativo à direita. pode encadea-lo.

```
const nota = 45
```

```
const conceito =
```

```
nota >70
```

```
? “Excelente”
```

```
: nota > 50
```

```
?”Médio”
```

```
: nota > 40
```

```
? “Ruim”
```

```
: “Péssimo”
```

Vetores (conj ordenado de dados) - tipados e dinâmicos
inicializar:

```
let vazio = [] //criado
```

```
let preenchido = [1, “”bola, ..., true] //vetor pré preenchido
```

```
let novo = new Array(); //vetor vazio usando operador new
```

```
let cabemDez = new Array(10);
```

```
let preenchido2 = new Array(1.”bola”, ..., true) //criado
```

preenchido2[100] = “novo elemento”

Principais métodos: length concat() reverse() slice() sort() splice() join() push() pop() unshift() shift()

Estruturas de repetição

while (condição) { Bloco de código que será executado }	do { Bloco de código pode ser executado } while (condição);	for (let i = 0; i < 10; i++) { console.log(`\${i}, "); //0,1,2,3,4,5,6,7,8,9 }
	A condição é testada após a execução do bloco, o laço é executado pelo menos uma vez.	const v = [1, 2, 3, 4, 5] for (let i in v) { console.log(`v[\${i}] = \${v[i]}`) } <small>ps: usado p/ filtrar elementos, transformar vetores ou resumir o vetor em um resultado</small>

Alterando fluxo normal de um laço: break(laço/switch mais interno abandonado), continue(iteração atual terminada, vá p/ prox iteração), return(retorna valor resultado da função), throw

Exceções (sinal que indica que ocorreu algum erro em tempo de execução) disparar erro usando throw, capturar para tratar Capturando com: try...catch...finally combo handles errors without stopping JavaScript. When an **error** occurs, JS will **stop** and generate an error message.

try defines the code block to run (to try). Que pode gerar a exceção

catch(err) defines a code block to handle any error.

err localreference to the error obj

finally defines a code block to run always, regardless of the result

throw statement defines a custom error.

catch and finally are optional, but you must use one of them.

```
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="m"></p>

<script>
function myFunction()
  let x = document.getElementById("demo").value;
  try {
    if(x == "") throw "is Empty";
    if(isNaN(x)) throw "not a number";
    if(x > 10) throw "too high";
    if(x < 5) throw "too low";
  }
  catch(err) {
    m.innerHTML = "Input " + err; //sets HTML of elem m to that string
  }
}
finally {
  document.getElementById("demo").value = "digit input here";
```

```
}
</script>
```

Objetos(conj ã ordenado de propriedades) são dinâmicos maperiam propriedades(strings) em valores, mapeamento chamado de tabela de hash/dicionário (???) Sintaxe obj literais (lista de nome:valor separadas por , dentro de {})

```
let vazio = { };
let ponto = { x: 0, y: 0 }
let pikachu = { nome: "Pikachu" , especie: "Pikachu" , nivel: 1, };
let livro = { titulo: "JavaScript", 'sub-titulo': "O Guia Definitivo", autor: { nome: "David", sobrenome: "Flanagan" } };
```

Criação de obj usando new, após a craaição é possível adicionar metodos e atributos

```
let charmander = new Object(); charmander.nome = "Charmander";
charmander.especie = "Charmander"; charmander.nivel = 5;
```

Acessando propriedades: 2 notações . and []

```
let pikachu = { nome: "Pikachu", ... }
let charmander = new Object(); //criou
charmander.nome = “Charmander”; //adicionou valora acessando
console.log(pikachu.nome); //acessar
console.log(charmander['nome']);//acessar
```

Bracket notation, Permite acessar propriedade cujo nome são calculados em execução. Não é preciso conhecer o nome previamente

```
let carteira = { btc: 0.009, eth: 0.195, bnb: 0.18, ... ... }
for (let cripto in carteira) {
  console.log(` ${cripto.toUpperCase()}= ${carteira[cripto]} ` ) }
```

Verificando a existência de uma propriedade

```
let livro = { titulo: "JavaScript", autor: { nome: "David", sobrenome: "Flanagan" } };
console.log(livro.subtitulo) // undefined console.log(livro.subtitulo.length) //
TypeError
let tamanho = undefined if (livro) { if (livro.titulo) { tamanho = livro.titulo.length } }
tamanho = livro && livro.titulo && livro.titulo.length
```

Callback

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

<script>
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some; }

function myFirst() {
  myDisplayer("Hello"); }

function mySecond() {
  myDisplayer("Goodbye"); }

mySecond();
```

```
myFirst();
</script>

</body>
</html> //output of code: hello
```

Asynchronous programming, this way JScan start long-running tasks, and continue running other tasks in parallel.

```
function myCal(num1, num2, myCallback) //to use a function inside another function you have to put her in the arguments ps: remember not to use parenthesis.
setTimeout(functionName, 3000) //will do function in 3s
setInterval(myFunction, 1000); //specify a callback function to be executed for each interval

//instead of passing the name of a function as an argument to another function, you can always pass a function with their arg:
setTimeout(function() { myFunction("I love You !!!"); }, 3000);//the inner function with an argument too// anonymous function calls myFunction
```

But, asynchronus programmes are difficult to write and difficult to debug.

Because of this, most modern asynchronous JS methods don't use callbacks, they use **Promises** instead.

Promises

Um obj que representa a eventual completude ou falha de uma operação assíncrona. Usado pois JS é uma linguagem single thread (ñ é possível realizar + de uma operação ao mesmo tempo), no navegar ela compartilha um thread com diversas funções do próprio navegador Interagir sem depender do backend no navegador Registra consumidores usando then e catch (em caso de falha), ambos tem retornar algum valor se não as callbacks ã terão acesso ao resultado

```
//contain producing code and calls to the consuming code:
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject(); // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
); //you can add a callback for success or failure only instead as well
```

myPromisse.state	myPromisse.result	
pending	undefined	inicial, nem rejeitada nem resolvida
fulfilled	a result value	sucesso
rejected	an error obj	rejeitada
setted		final, indica que foi resolvida ou rejeitada

How to use a promise

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

let myPromise = new Promise(function(myResolve, myReject) {
  let x = 0; // The producing code (this may take some time)
  if (x == 0) { myResolve("OK"); }
  else { myReject("Error"); }
});

myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
```

Ex Waiting a timeout

<pre>setTimeout(function() { myFunction("I love You !!!"); }, 3000); function myFunction(value) { document.getElementById("demo"). innerHTML = value; }</pre>	<pre>let myPromise = new Promise (function(myResolve, myReject) { setTimeout(function() { myResolve("I love You !!!"); }, 3000); }); myPromise.then(function(value) { document.getElementById("demo").innerHTML = value; });</pre>
---	---

Ex Waiting for a file

```
function getFile(myCallback) {
  let req = new XMLHttpRequest();
  req.open('GET', "mycar.html");
  req.onload = function() {
    if (req.status == 200) {
      myCallback(req.responseText);
    } else {
      myCallback("Error: " + req.status);
    }
  }
  req.send();
}

getFile(myDisplayer);

let myPromise = new Promise( function(myResolve, myReject) { //this function get's
executed immediately when Promise is created, receives 2 functions as arg
let req = new XMLHttpRequest();
req.open("GET", "mycar.html");
req.onload = function() { //function that gets called when the request completes.
  if (req.status == 200) {
    myResolve(req.response);
  } else {
    myReject("File not Found");
  }
};
req.send();
});

myPromise.then(//methods says what to do when promise is settled
  function(value) {myDisplayer(value);},//if Prom ok, then the resolved value is passed
to this anonymous function as the argument value
  function(error) {myDisplayer(error);}
);
```

async keyword before a function makes the function return a promise:

<pre>async function myFunction() { return "Hello"; }</pre>	
--	--

<pre>} myFunction().then(function(value) {myDisplayer(value);}, function(error) {myDisplayer(error);});</pre>	
<pre>async function myFunction() { return "Hello"; }</pre>	<pre>function myFunction() { return Promise.resolve("Hello"); }</pre>

await keyword can only be used inside an **async** function. **makes** the function pause the execution and wait for a resolved promise before it continues:

Promise funciona como ligação entre código produtor e o consumidor

<p>Em vez disso:</p> <pre>pagarComCartao(valor, callbackDeSucesso, callbackDeFalha)</pre> <p>Isso:</p> <pre>pagarComCartao(valor).then(callbackDeSucesso, callbackDeFalha)//pouco legível</pre>

Consumidores registrados pelos métodos then e catch

then: pega quando resolvida, **catch:** pega quando rejeitada -> ambos retornam algum valor (caso contrário as callbacks ã terão acesso ao resultado da promessa anterior)

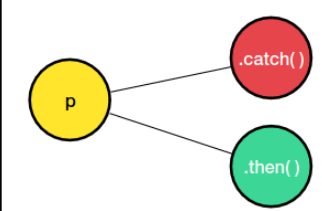
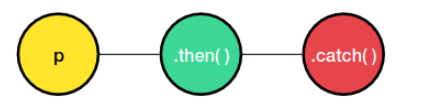
<pre>promise.then(r => console.log(r), e => console.log(e)) //2 parametros uma quando é resolvida outra quando rejeitada promise.then(r => console.log(r)) //só tenho interesse no sucesso promise.then(null, e => console.log(e)) //interesse só caso rejeitado promise.catch(e => console.log(e)) //rescrito assim</pre>
<pre>let promise = new Promise ((a,b) => { //in this new anonymous function 1° arg is resolve function, 2° reject function //some asynchronous operation setTimeout(() => a("operation is a success"), 1000); }); promise.then(result => { //1°arg result, so if successful will do result function console.log(result); });</pre>

Finally_



SEMPRE é executado
Geralmente é utilizado para “limpeza” após a execução da operação (Ex: Para indicadores de carregamento, liberar recursos não mais necessários)
Evita a duplicação de código dentro de blocos then e catch
Não possui argumento.

<pre>let loading = true const x = setInterval(() => { //1° arg: cria função anônima console.log(loading); if(!loading) clearInterval(x), 100) //if loading false, stops repeating const promise = new Promise((resolve, reject) => { //resolve and reject are both functions setTimeout(() => resolve("Deu bom"), 300) }) promise .then(r => console.log(r))//if successful, then r ="deu bom" .catch(r => console.log("Deu ruim")) // if rejected, then r would be whatever value was passed to reject() .finally(() => { loading = false; console.log("limpando")})</pre>	<pre>true Deu bom limpando false</pre>
---	--

Encadeada

<pre>const p = new Promise((resolve, reject) => { Math.random() > 0.5 ? resolve('yay') : reject('no') })</pre>	
<pre>p.then((res) => {})</pre>	<pre>p.then((res) => {}).catch((rej) => {})</pre>
<ul style="list-style-type: none">Cenário 1 - Dois <i>bindings</i> na mesma promise 	<ul style="list-style-type: none">Cenário 2 - Encadeamento entre o then e catchO catch será feito na promise retornada pelo then 

um catch p/ controlar todos

<pre>const p = new Promise((resolve, reject) => { return (Math.random() > 0.5) ? resolve('yay'): reject('no') }) p .then(function acao1 (res) { console.log(` \${res} da ação 1`); return res; }) .then(function acao2 (res) { console.log(` \${res} da ação 2`); return res; }) .then(function acao3 (res) { console.log(` \${res} da ação 3`); return res; }) .catch(function erro (rej) { console.error(rej) })</pre>
 

Independente do tratamento que damos a Promise, ele sempre vai buscar o primeiro tratamento de erros disponível

- Cada catch irá capturar o erro relativo às Promises anteriores
- Em seguida, o valor que ele retornar será passado para a próxima Promise que executará normalmente.
- O catch não é universal, quando encadeados em outros then, o primeiro erro que acontece consome o primeiro catch e assim por diante
- Tal comportamento é difícil de se replicar com callbacks, visto que cada callback só iria capturar os erros deu sua execução

Async antes de uma função sign que ela sempre vai retornar uma promise
Assim que lançar algo foi resolvida, se lançar exceção promise foi rejeitada.
Pode esperar uma ou + promises
Await faz o JS esperar até que a promise alcance o estado settled
função suspensa até resolução da promise
Nesse meio tempo outras tarefas são executadas

How JS references HTML elements - DOM (doc obj model)

by id (return elem with this id): `document.getElementById(id)`
by class (returns array-like collection of all elem in the doc with this class):
`document.getElementsByClassName("myClass");`
by tag name (returns array-like ""): `document.getEkementByTagNam("p");// All <p> elements`
by selector (return array-like ""): `document.querySelector(".myClass");// First element with class "myClass"`
by (returns a static NodeList [similar to an array] of all elements that match the specified CSS selector(s)) `document.querySelectorAll("div.myClass");// All <div> elements with class "myClass"`
b

A expressão `$ {}` é usada em JavaScript dentro de strings template literais, que são definidas por crases (``). Elas permitem a inserção de variáveis ou expressões dentro de uma string de forma mais fácil e legível. Além de variáveis, você pode incluir qualquer expressão JavaScript dentro de `$ {}`, como operações matemáticas ou chamadas de função.

Arrow function - allows write shorter function syntax

Higher order function - function that takes another function as arg

Callback - function that gets passed as arg (dif. waits for something to finish ex: waiting for a click event...) only then does it get called back and executed
ex:
`document.addEventListener("pressed", respondToKey(event));`
`function respondToKey(event){`

`console.log("Key pressed.")`
`}`
When that happens, we can get it to send us some information that it'll only know once the event happens (ex: which button/key was clicked)

Promise:

fetch (file)
`.then(x => x.text())` //x é obj resposta dessa promise
`.then(y =>`
`document.getElementById("demo").innerHTML = y);` // y
resultado da função text()

Incluindo JS - diretamente no arq HTML, tag `<script>`
`</script>` colocado no final do `<body>`
can use: `<script>` `<script async>` `<script defer>`
1. Navegador cria obj **Document** add nós de objs **Element** e **Text** propriedade **document.readyState** tem o valor **"loading"**
2. se só `<script>` adiciona esses elementos no documento e executa o script (executadas de forma sincrona)
3. Quando completamente analisado ->
document.readyState "interactive"
defer é executado na ordem que aparece no documento assincronos tb podem ser executados nesse momento
4. Navegador dispara um evento **DOMContentLoaded** no obj **Document**.
5.
6.
7.

DOM - doc obj model (modelo estruturado que o navegador constroi a partir do conteúdo)
Tipos de nós na DOM:

1. document.....

TypeScript

specifying the types of data and report errors when the types don't match.

```
let firstName: string = "Dylan";
let name = "Dylan"; // or doesn't specify type
let v: any = true;
v = "string"; // no error as it can be "any" type
let w: unknown = 1;
w = "string"; // sometimes error when it's unknown
let multiType: number | boolean; // pode ser qualquer um dos tipos
Union
let multiType: number | boolean;
type testResult = "pass" | "fail" | "incomplete";
let myResult: testResult;
myResult = "incomplete"; /* Valid
myResult = "pass"; /* Valid
myResult = "failure";
type dice = 1 | 2 | 3 | 4 | 5 | 6; //Type Aliases
let diceRoll: dice;
diceRoll = 1; /* Valid
diceRoll = 2; /* Valid
diceRoll = 7; /* Invalid
```

Arrays

```
const names: string[] = [];
names.push("Dylan");
```

```
const names: readonly string[] = ["Dylan"]; // prevent change in array
Tuples typed array
let ourTuple: [number, boolean, string]; // define our tuple, ps: only 3 elems
ourTuple.push('Something new and wrong'); // no type safety in for indexes 3+
```

```
Named tuples
const graph: [x: number, y: number] = [55.2, 41.3];
Delete tuples
const graph: [number, number] = [55.2, 41.3, 65.37];
const [x, , z] = graph; // destroyed 1° and 3° not 2° elem, order is import
```

```
Object
const car: { type: string, model: string, year: number } = { //no need
  type: "Toyota",
  model: "Corolla",
  year: 2009
};
const car: { type: string, mileage?: number } = { // no error
  type: "Toyota"
};
```

```
Index signatures
const nameAgeMap: { [index: string]: number } = {};
nameAgeMap.Jack = 25;
nameAgeMap.Mark = 5; // { Jack: 25, Mark: 5 }
Enum (group of constraints): can be numeric or string
enum CardinalDirections {
  North,
  East,
  South,
  West
```

```
}
let currentDirection = CardinalDirections.North; // 0

enum CardinalDirections {
  North = 1,
  East,
  South,
  West
};
console.log(CardinalDirections.West);// 4
```

```
enum StatusCodes {
  NotFound = 404,
  Success = 200,
  Accepted = 202,
  BadRequest = 400
}
console.log(StatusCodes.Success); //200
```

```
enum CardinalDirections {
  North = 'North',
  East = "East",
  South = "South",
  West = "West"
};
console.log(CardinalDirections.North); // "North"
Interfaces - type aliases for objects
interface Rectangle {
  height: number,
  width: number
}
```

```
const rectangle: Rectangle = {
  height: 20,
  width: 10
};
//can be extended
interface Rectangle {
  height: number,
  width: number
}
```

```
interface ColoredRectangle extends Rectangle {
  color: string
}
```

```
const coloredRectangle: ColoredRectangle = {
  height: 20,
  width: 10,
  color: "red"
};
Functions
function printStatusCode(code: string | number) {
  console.log(`My status code is ${code}.`)
} // UNION
```

```
let addNumbers = function (x: number, y: number): number {
```

```
return x + y;
}
let addNumbers1 = (x: number, y: number): number => x + y;
function addNumbers2 (x: number, y?: number): number { //y is optional
  return x + (y || 1);
}
function addNumbers3 (x: number, y = 25): number {
  return x + y;
}
//Rest parameters - array parameter
function add(a: number, b: number, ...rest: number[]) {
  return a + b + rest.reduce((p, c) => p + c, 0);
} //rest of numbers entered are gonna be the array
Change the type of a variable - AS
let x: unknown = 'hello';
console.log((x as string).length);
<>
let x: unknown = 'hello';
console.log((<string>x).length);
Force casting - 1° cast to unknown then to the target type
Classes: + types and visibility modifiers
3 main visibility modifiers: public, private, protected
class Person {
  private name: string;

  public constructor(name: string) {
    this.name = name;
  }
}
```

Node.js

open source server environment, uses JS on the server
asynchronous programming! eliminates waiting
cross-plataform runtime p/ criar aplicações server-side em JS (executada no SO, ã navegador)
prover suporte a API + tradicionais dos SO (HTTP, FileSystem)
usa V8 ou Chrome V8, engine open-source JS
-> Compila e executa código JS, gerencia a alocação de memória
e realiza a desalocação de objetos não necessário
WHAT IS IN A NODE.JS FILE?
tasks executed on certain events
file must be initiated on the server to have an effect
file has .js extension

```
INCLUDE MODULES: use require() function, arg is the name of the module
ex: require('http'); //now with access to http module, create server
      http.createServer(function (req, res) {
        res.writeHead(200, {'Content-Type': 'text/html'});
        res.end("Hello World!");
      }).listen(8080);
```

```
CREATE YOUR OWN MODULE: use exports to make properties and methods
avaleable outside module file
exports.myDateTime = function () {
  return Date();
}; //if you save in a file called firtmodule.js
```



```
//you can now do this:
var http = require('http');
var dt = require('./myfirstmodule');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("The date and time are currently: " + dt.myDateTime());
  res.end();
}).listen(8080);
```

API

mediador entre aplicações

REST (REpresentational State Transfe, tipo de api)

Estilo arquitetural para sist distribuídos de hipermidia

6 princípios da arquitetura: Cliente-Servidor, Interface Uniforme, Sistema em camadas, Cache, Stateless, Code-on-demand

Cientes usam APIs para se comunicar com servidores (assim obtendo acesso aos seus recursos)

Rest utiliza o get, post, put delete do http

get -> requisição, post -> enviar informações que devem ser adicionadas ex:num banco de dados, put -> atualizar as informações

- É stateless, toda requisição ã é armazenada
- Usa sintaxe universal para identificar os recursos (URIs)
- tem content-types

VUE.JS

framework progressivo de js (pode usar numa parte da sua aplicação ou mais) para construção de interfaces de usuário.

Tratam eventos do usuário e envia-os ao viewModel

View ã mantém o estado da aplicação

Transforma informações do Model p/ View passando comandos da View para o Model

In normal JavaScript we need to write **HOW** HTML and JavaScript is connected, but in Vue we simply need to make sure that there **IS** a connection and let Vue take care of the rest.

Template-based syntax, two-way data binding, and a centralized state management.

2 APIs possible: composition or options

```
<body>

<div id="app"> //HTML that Vue is connected to
  {{ message }} // {{ }} placeholder for data, TEXT Interpolation
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script> //link to vue

<script> // vue instance, can contain data, methods...
const app = Vue.createApp({
```

```
  data() {
    return {
      message: "Hello World!"
    }
  }
})

app.mount('#app') // connecting to HTML

</script>
</body>
</html>
```

vue directives	
v-bind	Connects an attribute in an HTML tag to a data variable inside the Vue instance.
sintaxe:	<div v-bind:[attribute]="[Vue data]"></div> ex: //bind HTML attribute to data in the Vue instance
	<div id="app"> <div v-bind:class="vueClass"></div> // class:vueClass = pinkBG </div> <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script> <script> const app = Vue.createApp({ data() { return { vueClass: "pinkBG" } } }) app.mount('#app') </script>
css bind	<div v-bind:style="{ fontSize: size + 'px' }"> // size is a vue property Text example </div> <div v-bind:style="{ backgroundColor: isImportant ? 'red' : 'lightgray' }"> Conditional background color </div> // isImportant is a vue property
.myClass in css isImp is a bool in vue	shorthand for 'v-bind:' is simply ':' ex: <div :class="{impClass: isImp}">
v-if	Creates HTML tags depending on a condition. Directives v-else-if and v-else are used together with the v-if directive.
v-show	Specifies if an HTML element should be visible or not depending on a condition.
v-for	Creates a list of tags based on an array in the Vue instance using a for-loop.
v-on	Connects an event on an HTML tag to a JavaScript expression or a Vue instance method. We can also define more specifically how our page should react to a certain event by using event-modifiers.
v-model	Used in HTML forms with tags like <form>, <input>

	and <button>. Creates a two way binding between an input element and a Vue instance data property.
--	--

Model: dados da aplicação, armazenados como

ViewModel: camada intermediária que une a Model e a View. Utiliza o objeto Vue para conectar os dados as DOM

Data binding: dorma declarativa de vincular dados do model á view usando diretivas como v-bind e v-model (sincronização bidirecional)

Componentes: blocos reutilizáveis que encapsulam _

Diretivas: atributos especiais que estendem a funcionalidade HTML. (V-BIND, V-MODEL, V-IF, V-FOR)

principais aspectos

diretivas

componentes

criando uma aplicação em Vue
const app = Vue.createApp({ //criação de instancia de obj /* options */ }).mount('#app') //aplicação tem que ser montada a um elem DOM
const RootComponent = { /* options */ } //componente raíz (ponto inicial de renderização) Vue.createApp(RootComponent) .component('SearchInput', SearchInputComponent) .directive('focus', FocusDirective) .use(LocalePlugin) .mount('#app')

Pinia: controlar estado (flux?)

	seg	terça	quarta	quinta	sexta
8h-10h	web (B1L2)	web (B1L2)	eda (B3L4)	cg (B1L4)	cg (B1L4)
10h-12h	eda (B4S2)	mc (B2S1)	mc (B2S1)	redes (B3S1)	redes (B3S1)
12h - 13:30	-----	-----	-----	-----	-----
13:30 - 15:30	web	web	est (B4S2)	est (B4S2)	est.
15:30 - 17:30	cg	cg	redes	des (B3SD)	des (B3SD)
17:30-18	janta	janta	chegar em casa	chegar em casa	chegar em casa
18-20	cg	cg	est.	est.	est.
20-22	livre	livre	*livre	livre	*livre
	6h	6h	3:30	2h	4h

*Quarta :se arrumar 19:30 volêi(20-22h)	*curso (programar na semana) 3:30 de aulas, 1h para fazer exerc.
*Sexta: se arrumar 20:15 vôlei(20:30-22:30)	*est: alguma matéria OU curso OU prática OU udemy

probabilidade de ter sangue b ou rh-
tabela , incluo o b e rh-? sim, inclui essa situação

$$(b|rh- = b_com_rh-/n^\circ_de\ rh-_existentes$$

0,24 p/ diminuir 0,249