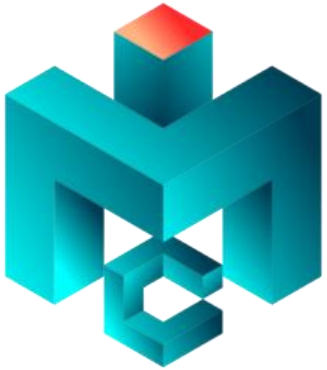


CONSULTA

Disciplina: XDES03 – Programação Web

Prof: Phyllipe Lima Francisco
phyllipe@unifei.edu.br

Universidade Federal de Itajubá – UNIFEI
IMC – Instituto de Matemática e Computação

HTML diz ao navegador onde e **quais elementos** estão na tela:

- ❑ Botões
- ❑ Formulários
- ❑ Tabelas
- ❑ Títulos
- ❑ Cabeçalhos
- ❑ Rodapé

HTML recomenda pensarmos no **significado** do elemento ou na área da página.

- ❑ Barra de Navegação <nav>
- ❑ Cabeçalho <header>
- ❑ Seções <section>
- ❑ Rodapé <footer>
- ❑ Parte principal <main>

CSS

CSS diz ao navegador como os elementos serão exibidos. Estilização

- Cor
- Margens
- Animações
- Cor de fundo
- Fonte
- Ordem

Como aplicamos a estilização?

- ❑ Criando regras
- ❑ Seletores
- ❑ Condição de corrida
- ❑ Modelo de Caixa (Box Model)
- ❑ Flexbox

JavaScript

JS permite adicionar <u>comportamento</u> nos elementos exibidos no navegador.

- ❑Tratamento de Eventos
- ❑Clique em botões
- ❑Envio/Recebimento de requisições
- ❑Submissão de formulários
- ❑Tratamento de erros
- ❑Manipular o DOM

Navegador

Navegador é capaz de exibir páginas web interpretando instruções HTML, CSS e JavaScript

HTML

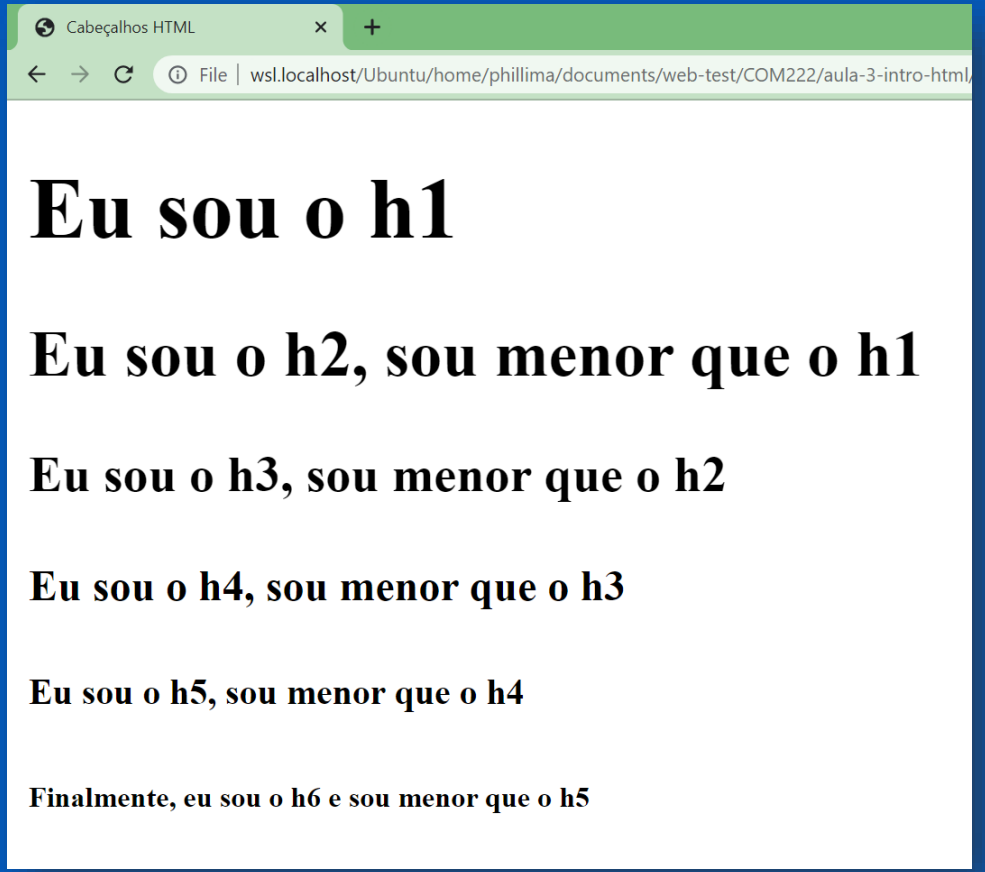
HTML

HTML utiliza etiquetas para marcar o conteúdo de uma página web.

<ETIQUETA>
Conteúdo
</ETIQUETA>

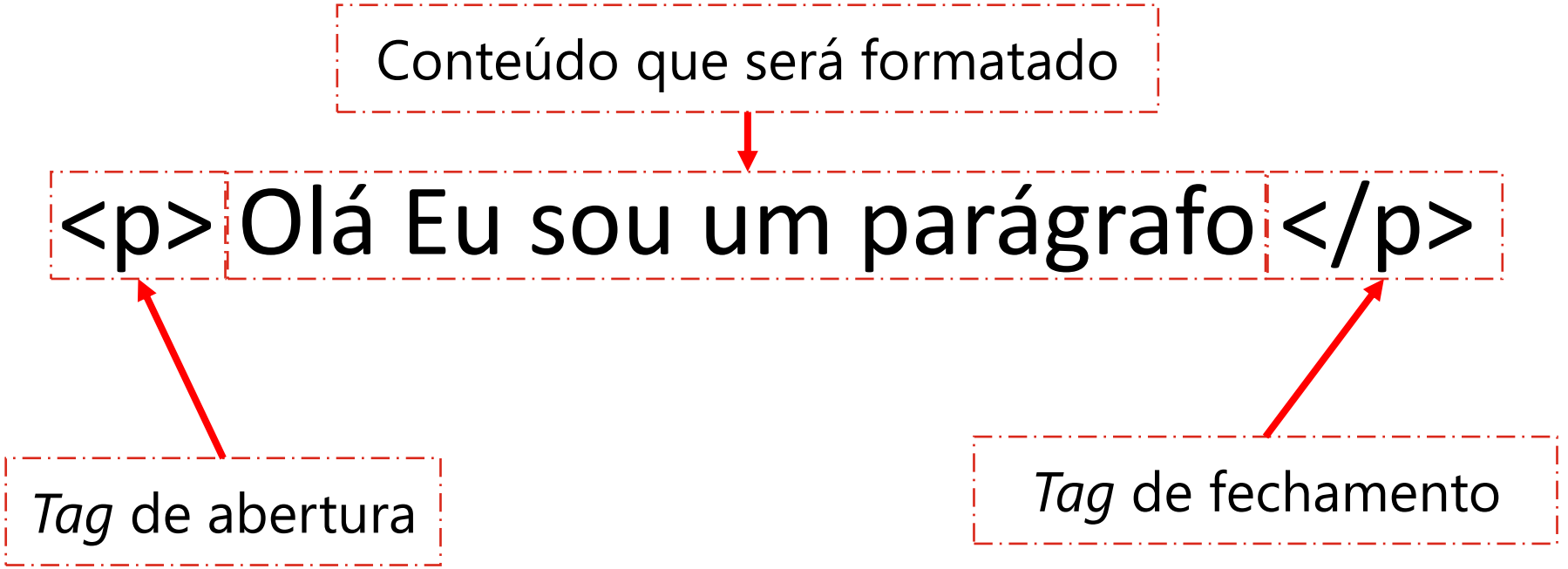
Código HTML com cabeçalhos

```
<h1>Eu sou o h1</h1>
<h2>Eu sou o h2, sou menor que o h1</h2>
<h3>Eu sou o h3, sou menor que o h2</h3>
<h4>Eu sou o h4, sou menor que o h3</h4>
<h5>Eu sou o h5, sou menor que o h4</h5>
<h6>Finalmente, eu sou o h6 e
sou menor que o h5</h6>
```



Parágrafo em HTML

- Podemos definir um parágrafo com a *tag* <p>.



Cabeçalhos e a semântica

- Não devemos utilizar os cabeçalhos apenas para aumentar/diminuir a fonte.
- Devemos fazer o uso consciente destes, nos preocupando com o real significado do conteúdo que as *tags* irão envolver.
- Uma boa regra é utilizar apenas um único <h1> por página. Os demais devem ser utilizados de forma a representar conteúdo hierárquicos.

HTML5 e a *tag* <HTML>

```
1. <!DOCTYPE html>
2. <html lang="pt-BR">
3. <head>
4.
   </head>
5. <body>
6.
7. </body>
8. </html>
```

Informa a versão do HTML

Tag de abertura da página HTML. O atributo "lang" está informando que é uma página em português do Brasil.

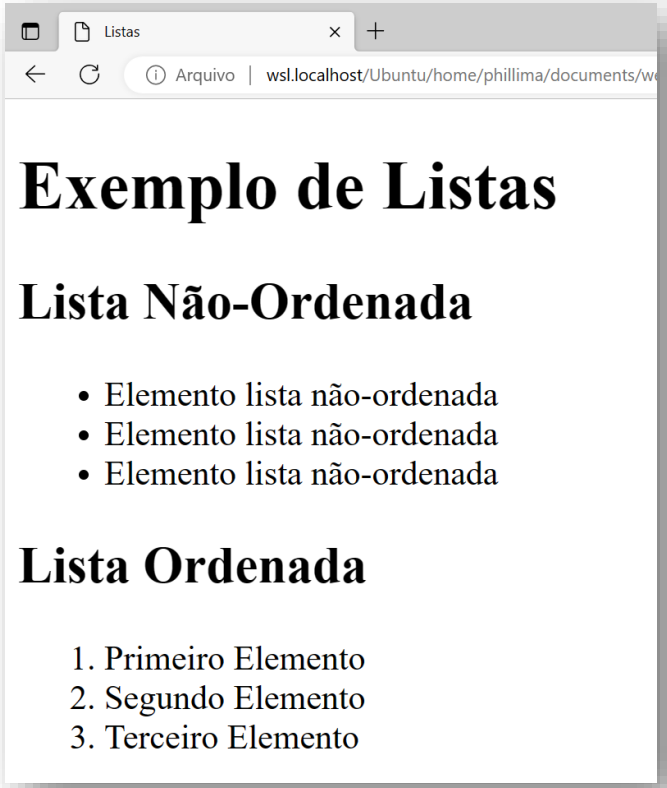
Tag de fechamento da página HTML

Listas com HTML

- Para listas ordenadas usamos a tag
- Para listas não-ordenadas usamos a tag
- Em ambas, definimos cada elemento com a tag

Listas com HTML – Código

```
<body>
    <h1>Exemplo de Listas</h1>
    <h2>Lista Não-Ordenada</h2>
    <ul>
        <li>Elemento lista não-ordenada</li>
        <li>Elemento lista não-ordenada</li>
        <li>Elemento lista não-ordenada</li>
    </ul>
    <h2>Lista Ordenada</h2>
    <ol>
        <li>Primeiro Elemento</li>
        <li>Segundo Elemento</li>
        <li>Terceiro Elemento</li>
    </ol>
</body>
```



Listas Aninhadas - Código

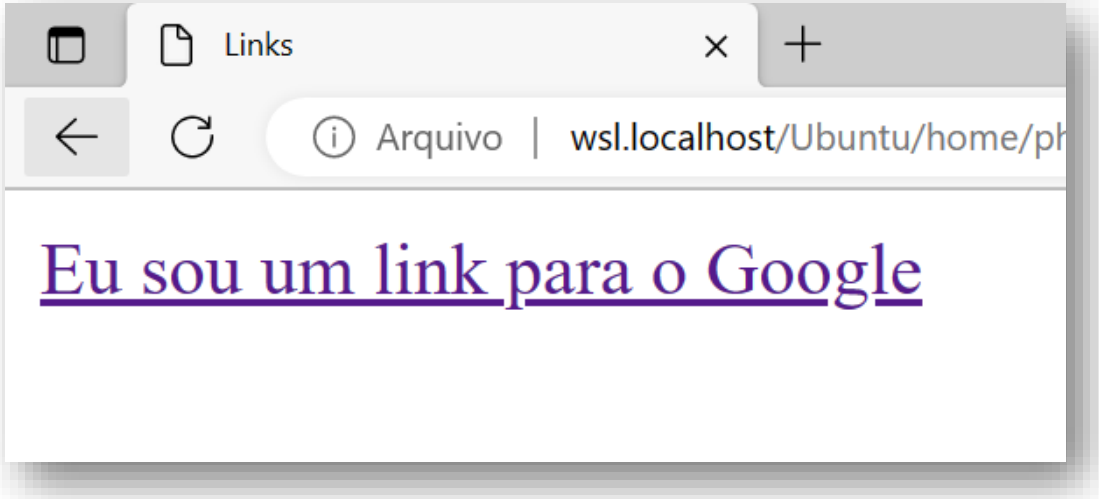
```
<body>
    <h1>Exemplo de Lista Aninhada</h1>
    <h2>Lista Ordenada dentro de uma Lista Não-Ordenada</h2>
    <ul>
         <li> Primeiro Item   <!-- Abertura tag <li> -->
            <ol>
                <li>Primeiro Elemento da Lista Aninhada</li>
                <li>Segundo Elemento da Lista Aninhada</li>
            </ol>
        </li>                  <!--  Fechamento da tag <li> ficou no fim -->
        <li>Outro item lista não-ordenada</li>
        <li>Último item lista não-ordenada</li>
    </ul>
</body>
```

Figuras com HTML – Código

```
<figure>
    <img src="figura.jpg" alt="descrição">
    <figcaption>Legenda</figcaption>
</figure>
```

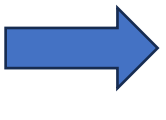
Criando hiperlinks em HTML - Código

- Eu sou um link para o Google
- Para abrir o link em outra aba, use o atributo "target" com valor "_blank"



Seleção - <select>

```
<select name="select-paises" id="paises">
        <option value="Argentina">Argentina</option>
        <option value="Bolívia">Bolívia</option>
        <option value="Brasil">Brasil</option>
        <option value="Chile">Chile</option>
        <option value="Colômbia">Colômbia</option>
        <option value="Equador">Equador</option>
</select>
```



- O campo "value" pode ser recuperado via JavaScript pela propriedade "value"

Elementos em Bloco

- Elementos em bloco, ocupam um bloco completo. Isto é, é reservado área em cima e embaixo.
- Elementos como <p>, <hx>, <li>, e demais são em blocos
- Podemos utilizar a tag para criar um bloco e agrupar elementos “em bloco”

Elementos em Linha (inline)

- Elementos em linha, ocupam apenas o espaço necessário para serem renderizados.
- Elementos como <a> são em linhas
- Podemos utilizar a tag para criar um bloco e agrupar elementos “em linha”

Semântica e HTML

O prato principal – a tag <main>

- A tag **<main>** deve incorporar o conteúdo principal.
- Tudo que é essencial e está diretamente relacionado ao conteúdo que se deseja apresentar.
- Barra de navegação, menu lateral, rodapé, e outros podem não fazer parte do conteúdo principal e portanto ficam fora da tag <main>

A tag <nav>

- A tag **<nav>** deve incorporar o conteúdo relacionado a barra de navegação com links para:
  - Páginas externas
  - Conteúdo na própria página
- Normalmente fica fora do conteúdo principal.
- Menus, índices de conteúdos, etc.

Exemplo com <nav>

- Wikipédia de Minas Gerais: https://pt.wikipedia.org/wiki/Minas_Gerais

A tag <section>

- A tag **<section>** representa um seção genérica de conteúdo em uma página. É usada quando não há uma tag mais específica.
- A tag**<section>** normalmente é acompanhada de um cabeçalho <h1>..<h6>

Exemplo - <figure>

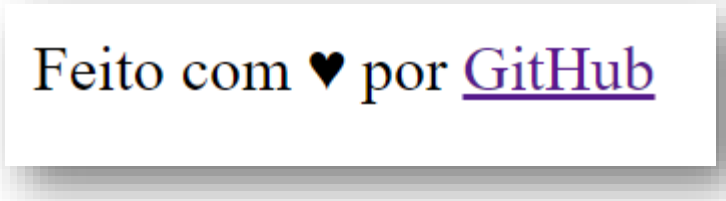
```
<figure>
    <img src="figure.png" alt="descrição da figura">
    <figcaption>Legenda para a Figura</figcaption>
</figure>
```

A tag <footer>

- A tag **<footer>** deve ser usada quando se deseja criar uma seção que se caracteriza como rodapé da página. Não usamos <section> pois temos uma tag mais específica.
- Normalmente fica fora da <main>

Exemplo tag <footer>

- “Feito com <3 por <sua empresa/pagina/etc>”.



A tag <header>

- A tag **<header>** deve ser usada quando se deseja introduzir um conteúdo.
- Pode ser utilizada como descendente de <nav>, <main>, <section> e outros.
- Pode conter informações de autoria, data/hora, imagem e outros.

Formulários <forms>

O que a tag <form> oferece?

- Com a tag <form> conseguimos usar atributos especiais que definem a ação e para onde as informações serão submetidas.
- Dentro da tag <form>, colocamos outras tags que irão, visualmente, compor o formulário como:
  - Entradas de texto, botões, rótulos, e etc.

Página com <form>

- Ao criamos uma página HTML com o seguinte formulário básico:

```
<form action="">


</form>
```

Entradas <input>

Entradas com a tag <input>

- A tag <input> permite uma grande variedade de tipos de entradas dentro de um formulário.
- Ao modificar o parâmetro “type”, o elemento se adapta para representar o tipo atribuído.

Entrada de texto com <input>

- O padrão é "entrada de texto"
- Pode ser colocado de forma explícita ao marcar "type" como "text"

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

Adicionando um *placeholder* (espaço reservado)

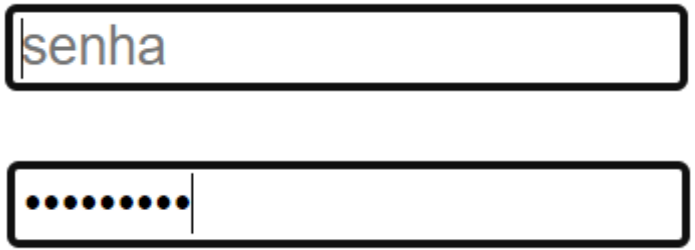
- Com o atributo placeholder, podemos deixar instruções para auxiliar o que deve ser colocado na entrada.

```
<input type="text" placeholder="nome de usuário">
```

Adaptando para entrada de senhas

- Modificando o atributo type para “password” o navegador irá ocultar o conteúdo sendo digitado.
- Isso pode auxiliar na proteção de senhas

```
<input type=“password" placeholder=“senha">
```



Rótulos com a tag <label>

- A tag <label> aprimora a legibilidade e acessibilidade de entradas em um formulário. Com ele é possível associar um rótulo a uma entrada.
- Para isso precisamos usar dois novos atributos
  - id no elemento <input>
  - for no elemento <label>

Associando rótulos as entradas

```
<label for="usuario">Nome de Usuário</label>
<input type="text" placeholder="usuário" id="usuario">
```

- Usando o atributo for em “label” especificamos “qual” entrada esse rótulo está associado.
- Na entrada, <input>, precisamos definir valor do atributo id como o mesmo colocador no “for”
- O valor de id precisa ser único na página.

Submetendo formulário com botões

- Para submeter informações de um formulário precisamos preencher o atributo action no <form> e adicionar um botão dentro do <form>
- A princípio, qualquer botão dentro do forma irá submeter os dados. Isto é, estará sujeito ao evento 'submit'.

Nomeando os parâmetros

- Ao submeter o formulário usamos o atribute "name" nos <inputs>.
- O atributo "name" é o nome do parâmetro que será recebido pelo servidor.

CSS

Sintaxe Básica

Sintaxe Básica

```
seletor {
   propriedade: valor;
}
```

Exemplo Sintaxe – Seletor Elemento

```
p {
   color: red;
}
```

Estilização Externa – Exemplo

- Arquivo de estilização styles.css

```css
p {
    color: red;
    font-size: 18px;
}

h1 {
    font-size: 32px;
    text-align: center;
    color: blue;
}
```

Seletores

Seletor de Elemento

- Seleciona todos os elementos do tipo especificado. O seletor é o nome da tag HTML.
- É o tipo de seletor que vimos nos exemplos anteriores.

Seletor de Classe

- Seleciona elementos com base na classe atribuída. Pode-se usar o seletor de classe para estilizar um grupo específico de elementos em sua página.
- Na sua definição é necessário colocar o caractere ponto (.)

Seletor de Classe – Exemplo definindo a classe

```
.minha-classe{
    text-align: center;
    color: blue;
    font-size: 22px;
}
```

Seletor de Classe – Exemplo usando a classe

```
    <section class="minha-classe">
        Eu sou seção e possuo o atributo class com valor
"minha-classe". Isso significa que todas regras de estilização
definidas em ".minha-classe" serão aplicadas em mim
    </section>
    <p class="minha-classe">Eu não sou uma seção, mas possuo o
atributo class com valor "minha-classe". Isso significa que
todas regras de estilização definidas em ".minha-classe" também
serão aplicadas em mim. Não é necessário sermos o mesmo
elemento HTML, mas termos o atributo da classe. Talvez isso não
seja adequado e possa gerar confusão, mas é possível.</p>
```

Seletor de ID (Identificador)

- No HTML, id é um atributo utilizado para definir um valor exclusivo para determinado elemento.
- O CSS pode utilizar esse valor exclusivo como um identificador. Se utiliza o "#" para fazer a seleção.
- Não devemos ter mais de um elemento HTML com o mesmo valor de id.

Seletor de ID (Identificador) – Exemplo CSS

```
#exemplo {
    color: green;
    background-color: black;
}
```

Seletor de ID (Identificador) – Exemplo HTML

```
<body>
    <h1>Eu sou um H1 e não tenho id.</h1>
    <p id="exemplo">Eu sou um parágrafo e tenho um
id. Lembrando que esse id não pode ser duplicado. Só
eu possuo esse valor</p>
</body>
```

Seletor de Atributo

- Seleciona elementos com base em um atributo específico. Podemos usar o seletor de atributo para estilizar elementos que possuem um determinado atributo ou valor de atributo.

Seletor de Atributo – Exemplo

- Estilizar ancoras <a> que redirecionam para o GitHub.

```
a[href="https://www.github.com"]{
    color:green;
}
```

Seletor de Atributo – Exemplo

- Estilizar parágrafos <p> que tenham um id

```
p[id]{
    color:green;
}
```

Color

- A propriedade cor modifica a cor do texto e pode receber valores em forma de:
  - Nome
  - RGB
  - RGBA
  - Hexadecimal

Background-color

- A propriedade background-cor modifica a cor do fundo e pode receber valores em forma de:
  - Nome
  - RGB
  - RGBA
  - Hexadecimal

Text-align

- Descreve como o texto é alinhado no elemento

  Pode assumir valores como:

  - left
  - right
  - center
  - ...

Font-weight

- Controla o peso da fonte. Pode receber valores nominais e numéricos.
- Pode variar conforme a fonte
  - normal
  - bold
  - 100

Text-decoration

- Pode ser quebrada em três propriedades:
- text-decoration-line
- text-decoration-style
- text-decoration-color
- É possível passar os três valores diretamente
  - text-decoration:

Text-decoration-line

- Define um tipo de linha para decorar o texto
  - none
  - underline
  - overline
  - line-through

Text-decoration-style

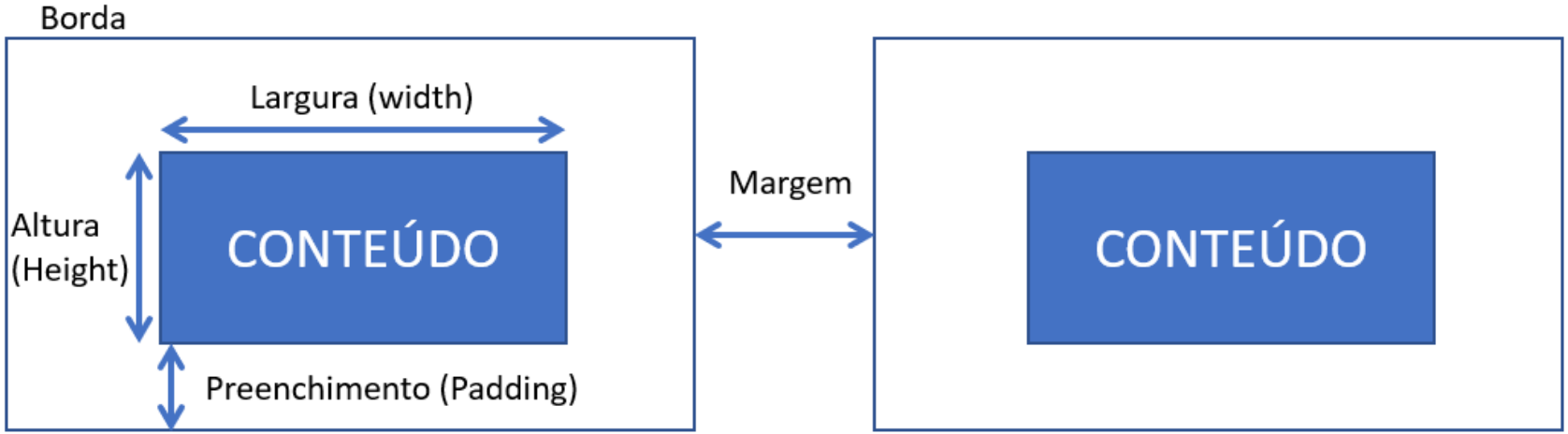
- Define um estilo para a linha decorando o texto
  - solid
  - double
  - dotted
  - wavy

Text-decoration-color

- Define a cor para a linha decorando o texto. Pode receber valores em forma de:
  - nome
  - RGB
  - RGBA
  - Hexa

Modelo Caixa

Modelo Caixa



Largura e Altura

- Podemos definir a altura (height) e largura (width) de um elemento atribuindo valores diretamente nessas propriedades
- Essas medidas irão definir a área retangular com o conteúdo. É o retângulo mais interno.

Largura e Altura - Exemplo

```
p{
    width: 200px;
    height: 200px;
}
```

Preenchimento

- Valor que será adicionado entre o conteúdo e a borda, isto é, o preenchimento fica dentro do elemento HTML

```
padding-top: ;
padding-right: ;
padding-bottom: ;
padding-left: ;
padding: ;
```

Margem

- Representa a distância externa a borda. É ela quem irá determinar qual distante o elemento ficará dos demais.
- Para controlar a margem usamos a propriedade “margin”.

Borda

- Representa o retângulo mais externo do elemento HTML.
- A princípio a borda fica invisível e precisa ser modificada para que se torne presente/visível.
- Pode ser estilizada de várias maneiras, incluindo largura, cor e estilo.

Borda

```
seletor{
    /* Coloca uma borda */
    border: 1px solid black;
     /*Arredondar a borda  */
    border-radius: 1em;
}
```

Medidas

Medidas Absolutas

- PX (CSS pixel): Unidade absoluta mais utilizada. A confusão feita é que 1px não é necessariamente 1 pixel no monitor, definido como a menor unidade endereçável. Não é recomendado para sites responsivos.

Medidas Relativas - Porcentagem

- % (porcentagem): Unidade relativa que ocupa o espaço em relação a uma porcentagem do elemento “pai”.

Medidas Relativas - EM

- EM: Unidade relativa ao tamanho da fonte aplicado no elemento "pai". Se o elemento pai possui uma fonte de 16px, e o filho uma fonte de "1em", este será 16px. Mas se for "2em" este será 32px. Conforme os objetos são aninhados, o valor de EM pode se alterar

Medidas Relativas - REM

- REM: Unidade relativa ao tamanho da fonte aplicado no elemento "raiz". Se o elemento raiz possui uma fonte de 16px, e o filho uma fonte de "1rem", este será 16px. Mas se for "2rem" este será 32px. Conforme os objetos são aninhados, o valor de REM não se altera.

Medida Relativa - VH

- VH -> View Height
- Medida Relativa a altura do display visível.
- Pode ser usada para definir altura relativa de outros elementos.

```
seletor{
   /* Altura será 10 unidades de VH */
   height: 10vh;
}
```

Display

A propriedade Display - Definição

- Usada para controlar a forma que um elemento HTML é exibido na página.
- Pode ser usada para mudar o comportamento padrão de um elemento e torná-lo mais flexível

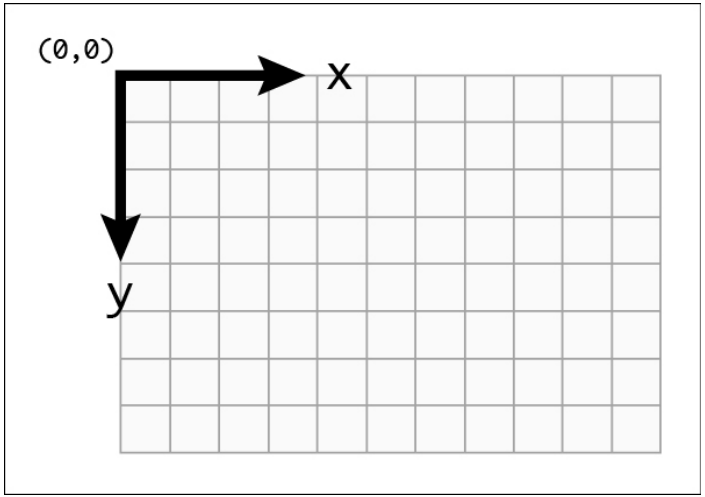
A propriedade Display - Valores

```
display: block ;
display: inline;
display: inline-block;
display: none;
display: flex;
```

A propriedade Display - Valores

- ***block*** => Respeita medidas como “height” e “width”
- ***inline*** => Ocupa apenas o espaço necessário
- ***inline-block*** => Ocupa apenas o espaço necessário, mas respeita “height” e “width”.
- ***flex*** => transforma em um contêiner flex

Posicionamento



A propriedade *Position* - Definição

- Permite definir como um elemento deve ser posicionado na página em relação aos outros elementos.

A propriedade *Position* - Valores

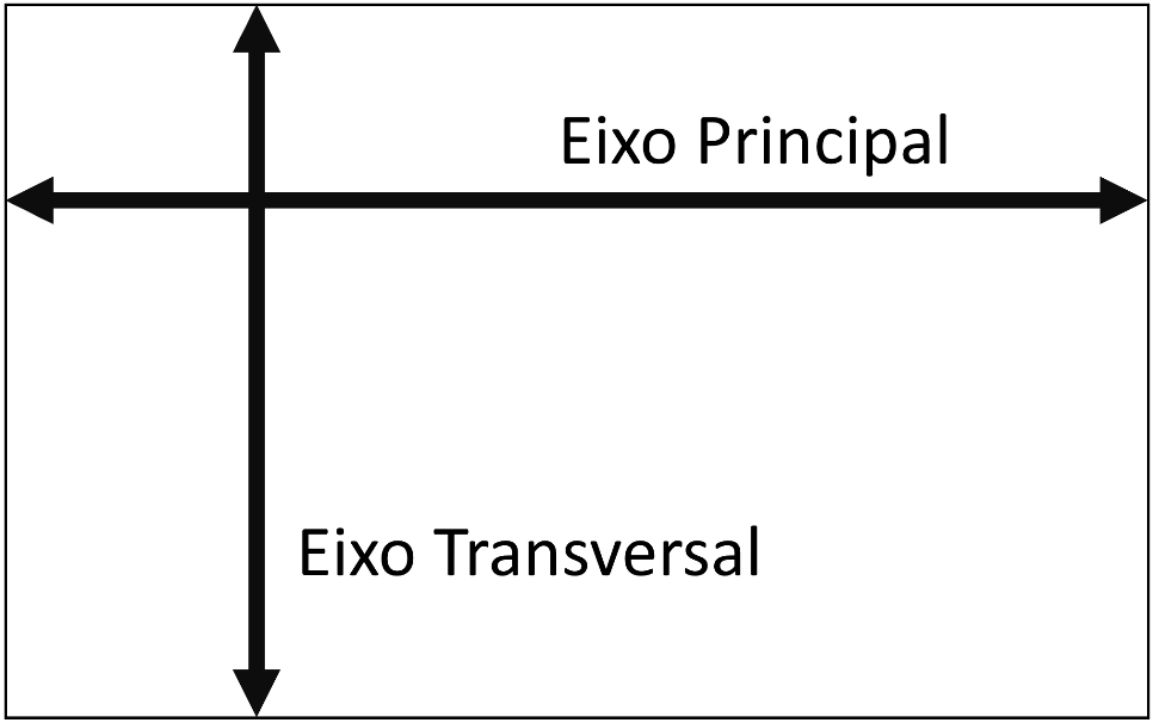
```
position: static;
position: relative;
position: absolute;
position: fixed;
position: sticky;
```

A propriedade Display - Valores

- ***static*** => Ocupa posição padrão
- ***relative*** => Ocupa posição em relação a sua posição padrão. A posição pode ser modificada com as propriedades “top” e “left”
- ***absolute*** => É removido do fluxo da página e pode ser posicionado usando “top” e “left”. A origem é o canto superior esquerdo.
- ***fixed*** => Ocupa uma posição fixa na tela
- ***sticky =>*** Semelhante ao fixed. Mas irá “grudar” (stick) na tela assim que a parte superior a encontrar

Modelo Flexbox

Modelo Flexbox



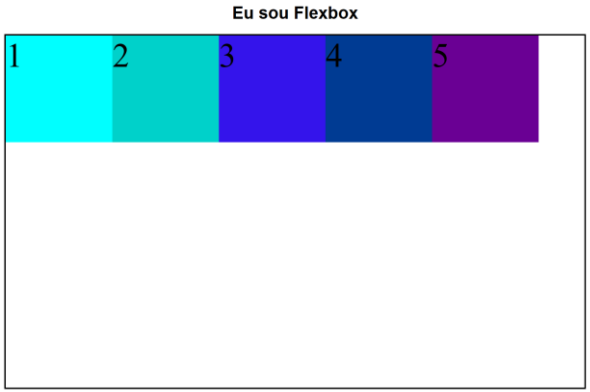
Eixos – Principal

- Ao declararmos um elemento com a propriedade "flex", por padrão, o conteúdo é posicionado em linha, da esquerda para a direita.
- A propriedade utilizada para esse controle é a "**flex-direction**".

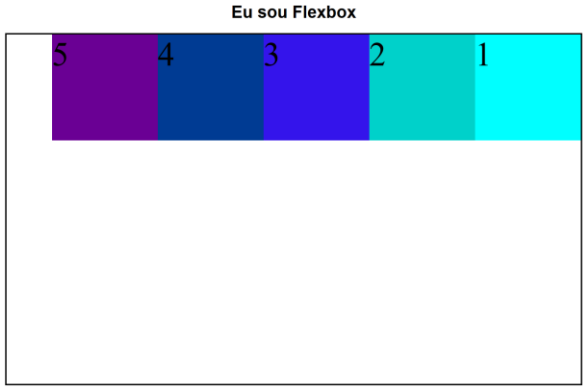
Eixos – Principal

```
/* Valor padrão. O eixo principal é horizontal, da esquerda para a direita */
flex-direction: row;

/* O eixo principal é horizontal, da direta para a esquerda */
flex-direction: row-reverse;

/* O eixo principal é vertical (coluna), de cima para baixo */
flex-direction: column;

/* O eixo principal é vertical (coluna), de baixo para cima */
flex-direction: column-reverse;
```

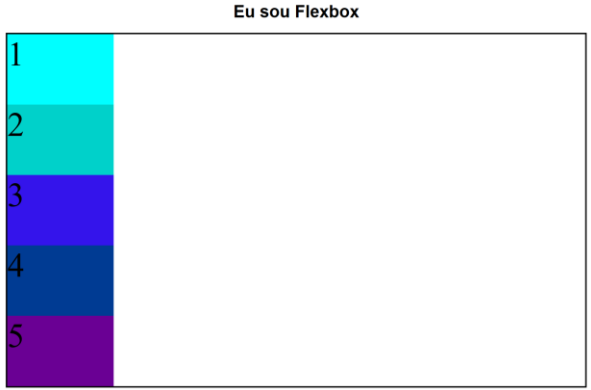
Eixos – Principal - Exemplos




```
flex-direction: row;
```




```
flex-direction: row-reverse;
```




```
flex-direction: column;
```

Alinhamento e Propriedades

Alinhamento no Eixo Principal

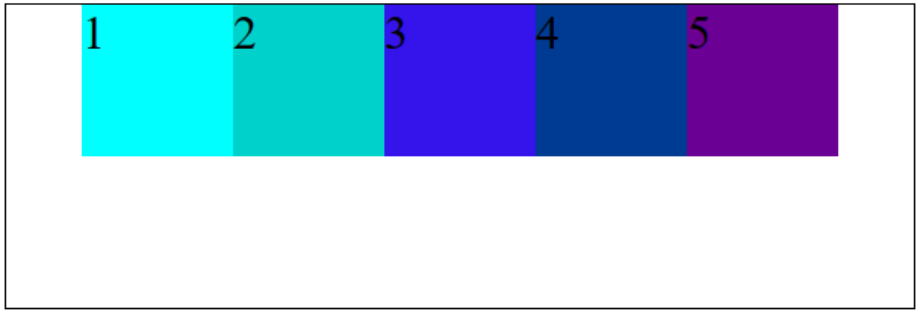
- Para alinhar os elementos no eixo principal, usamos a propriedade **justify-content**.
- Seu comportamento irá depender de como a propriedade **flex-direction**

Alinhamento no Eixo Principal : justify-content

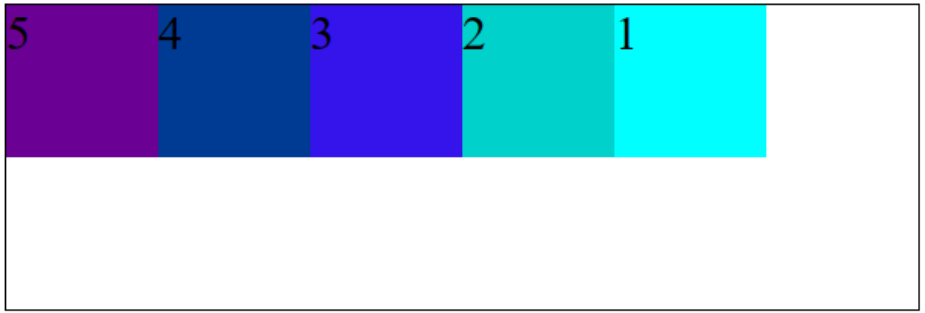
```
/* Alinhamento padrão, com os elementos posicionados de acordo com o
início do eixo principal  */
justify-content: flex-start;

/* Elementos posicionados de acordo com o fim do eixo principal  */
justify-content: flex-end;

/* Elementos centralizados no eixo principal  */
justify-content: center;

/* Elementos com o espaço distribuído ao redor  */
justify-content: space-around;

/* Elementos com o espaço distribuído entre os elementos  */
justify-content: space-between;

/* Elementos com o espaço distribuído igualmente entre os elementos  */
justify-content: space-evenly;
```

Alinhamento no Eixo Principal : justify-content

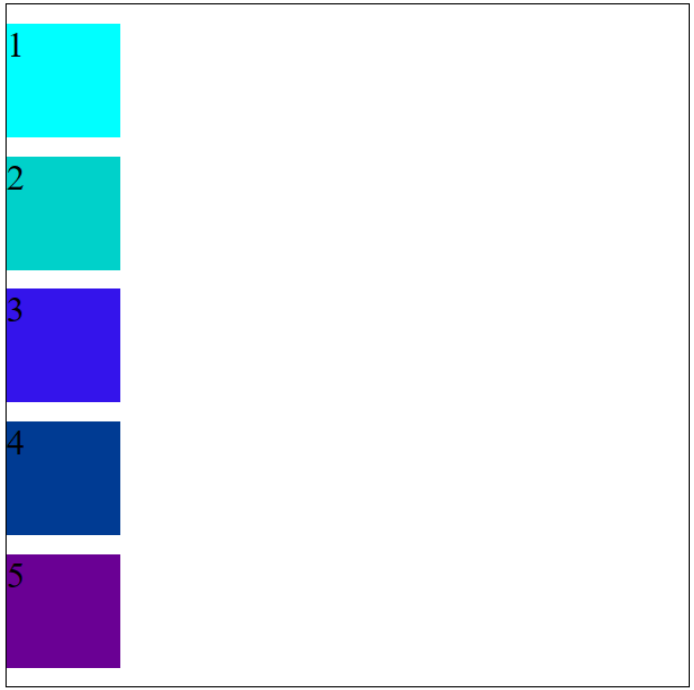
Row e Center



Row Reverse e Flex End



Column e Space Evenly



Wrap do Elementos

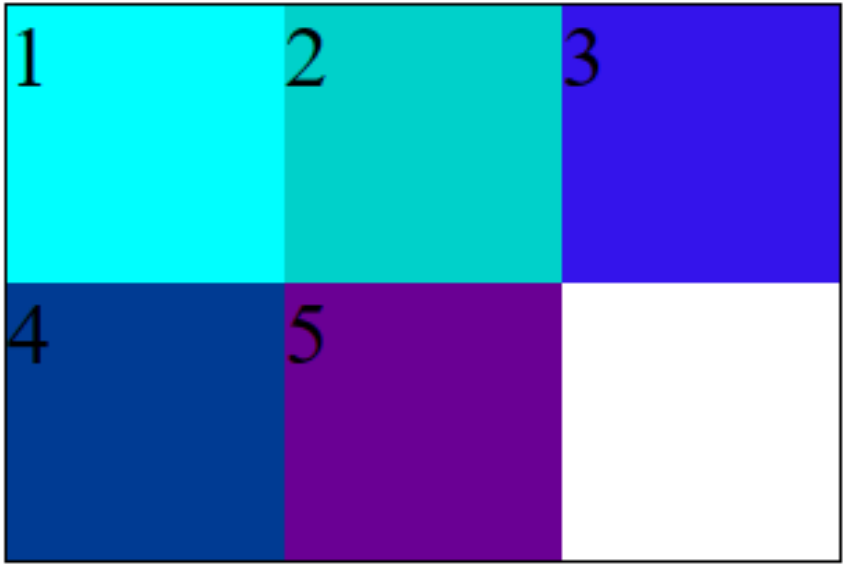
- A propriedade **flex-wrap** define se os elementos são forçados a ficarem na mesma linha ou se podem ser quebradas em várias linhas.
- O sentido que os elementos serão “quebrados” segue o eixo transversal.

Wrap do Elementos : flex-wrap

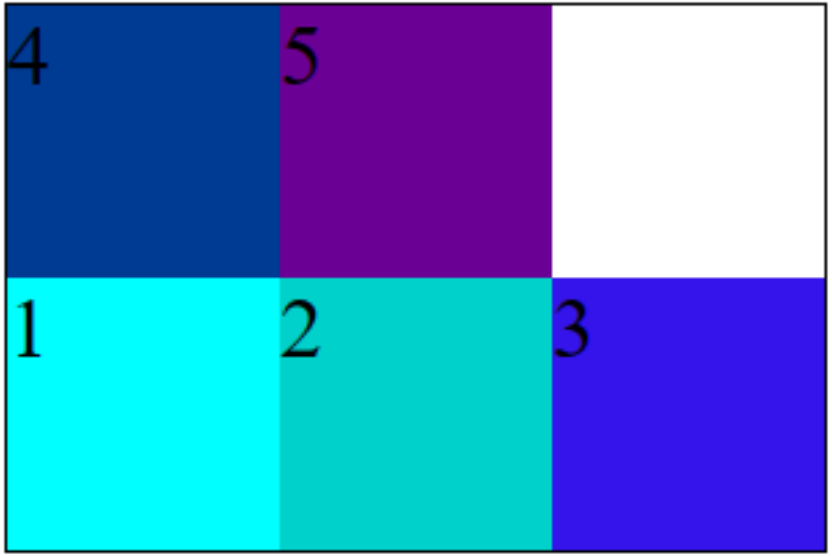
```
/* Valor padrão. Os elementos não passam para proxima linha/coluna */
flex-wrap:nowrap ;

/* Na mesma direção do eixo transversal */
flex-wrap: wrap;

/* Na direção contrária  ao eixo transversal
flex-wrap: wrap-reverse;
```

Wrap do Elementos : flex-wrap

Row e Wrap

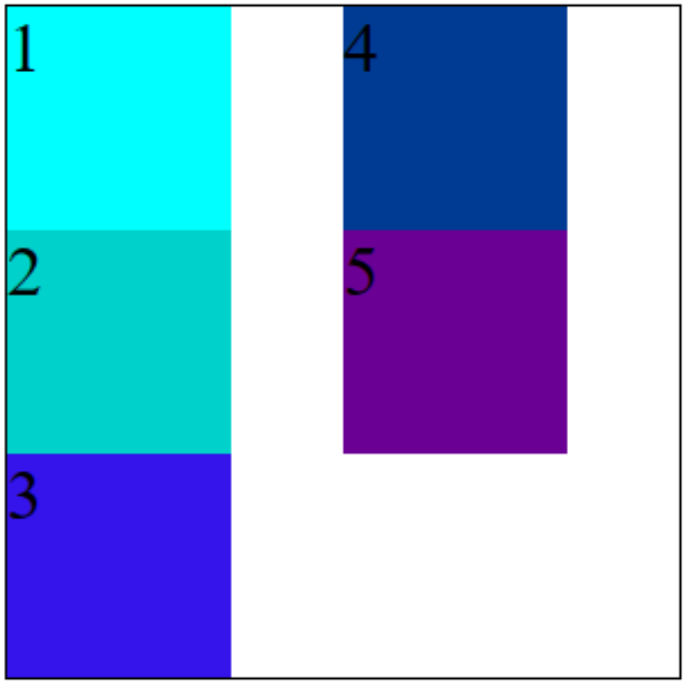


Row e Wrap-Reverse

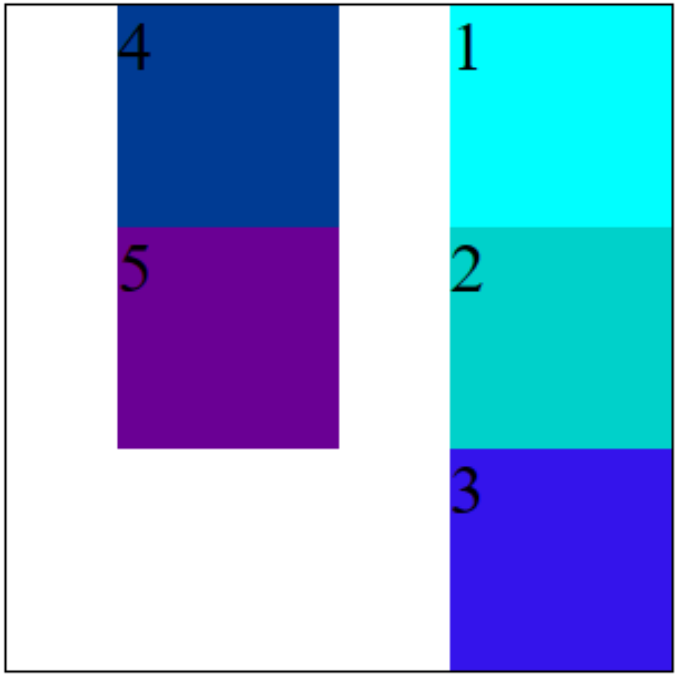


Wrap do Elementos : flex-wrap

Column e Wrap



Column e Wrap-Reverse



Alinhamento no Eixo Transversal

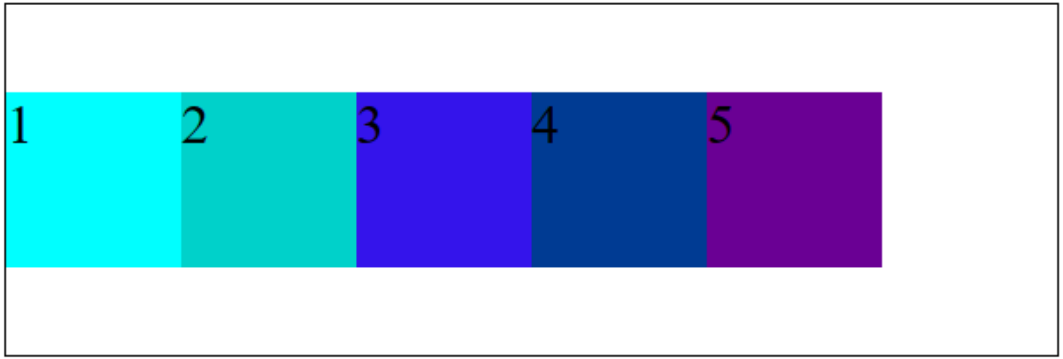
- Para alinhar os elementos de um “flex” container, ao longo do eixo transversal, usamos a propriedade **align-items**.
- A propriedade **justify-content**, alinha e espaça os elementos ao longo do eixo principal.

Alinhamento no Eixo Transversal: align-items

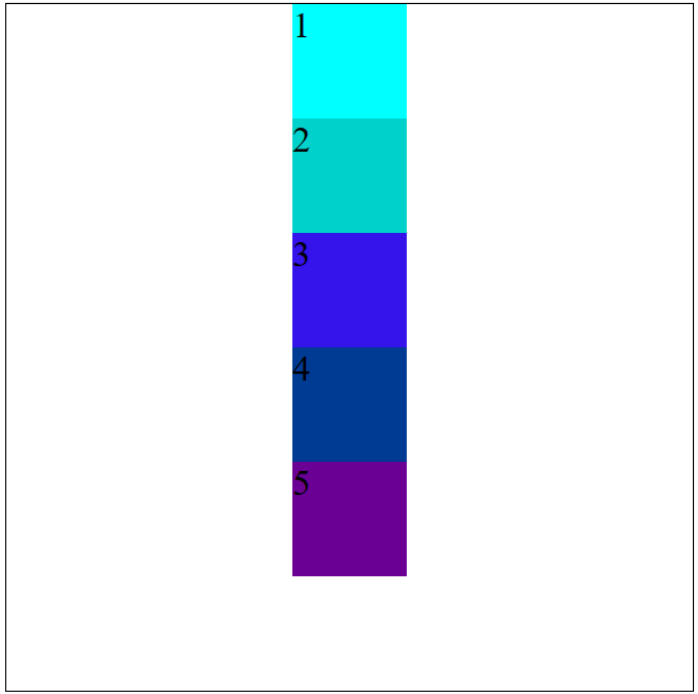
```
/* Valor padrão. Alinha os elementos a partir do início do eixo
transversal  */
align-items: flex-start;

/* Alinha os elementos a partir do fim do eixo transversal  */
align-items: flex-end;

/* Centraliza os elementos no eixo transversal  */
align-items: center;

/* Alinhas os elementos no eixo transversal de acordo com a base do
conteúdo  */
align-items: baseline;
```

Alinhamento no Eixo Transversal

Row e Center-Transversal



Column e Center-Transversal

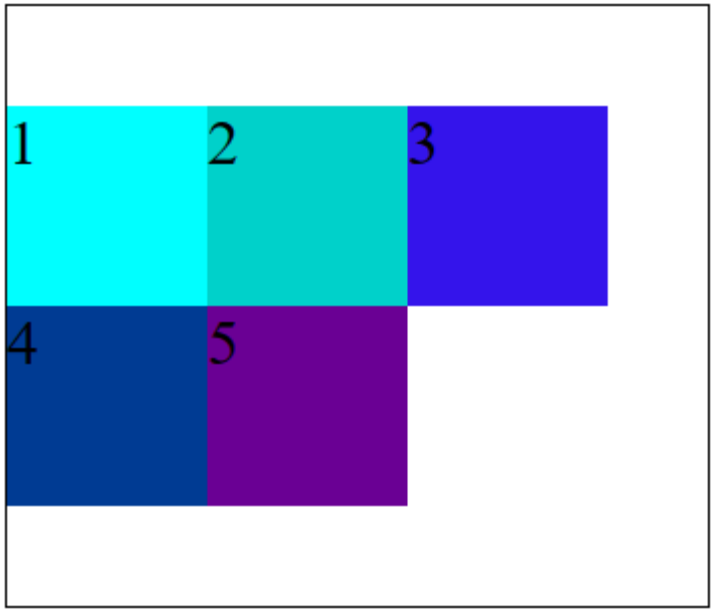


Alinhamento com Múltiplas Linhas/Colunas

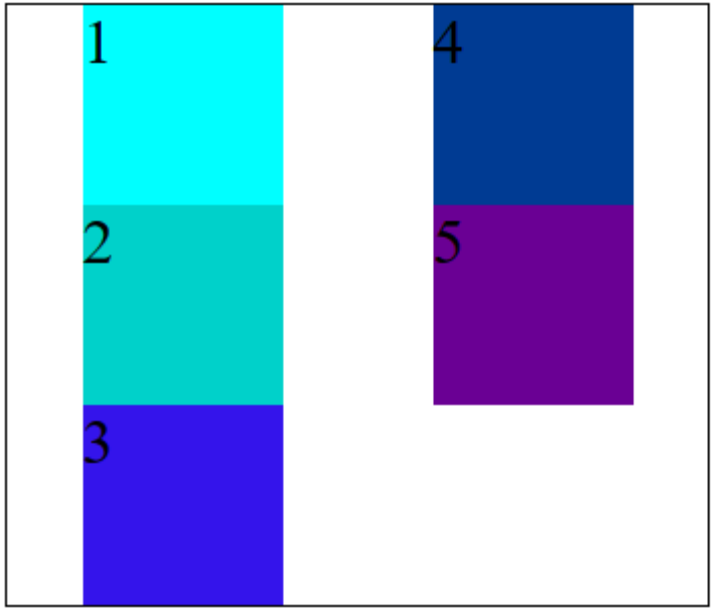
- Quando se tem múltiplas colunas e/ou linhas usamos a propriedades **align-content** para controlar o espaçamento.
- É necessário ter algum tipo de "wrap" ativado.

Alinhamento com Múltiplas Linhas/Colunas

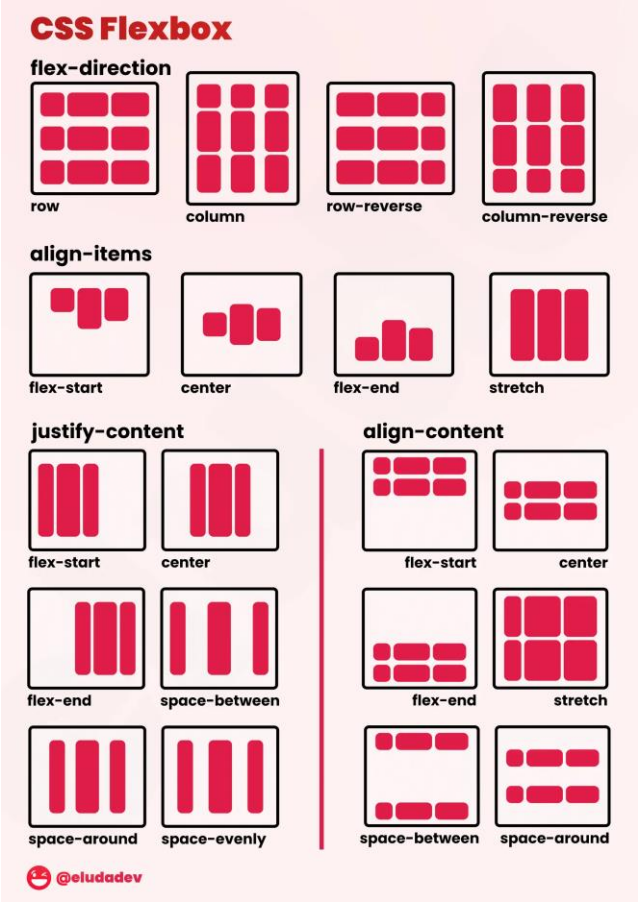
Row, Wrap e Align-Content: Center



Column, Wrap e Align-Content: Space-Around



Resumo Display Flex



Media Queries

Layout Responsivo

- Permite modificar a estilização dependendo de características como tamanho de tela e tipo de dispositivo.
- Podemos especificar uma largura, orientação, e outras condições para alterarmos o layout.

Media Queries – Exemplo 1

- Modificar a cor do elemento <h1> quando a tela estiver **exatamente** com 800px:

```
@media (width: 800px){
    h1{
        color: purple;
    }
}
```

Media Queries – Exemplo 2

- Modificar a cor do elemento <h1> enquanto a tela tiver **pelo menos** 800px;

```
@media (min-width: 800px){
    h1{
        color: purple;
    }
}
```

Media Queries – Exemplo 3

- Modificar a cor do elemento <h1> enquanto a tela tiver **entre** 600px e 800px;

```
@media (min-width: 600px) and (max-width:800px){
    h1{
        color: purple;
    }
}
```

Imagens

Object-Fit

- Especifica como o conteúdo de uma imagem/ elemento deve ser ajustado ào container.

```
object-fit: fill
object-fit: contain
object-fit: cover
object-fit: none
object-fit: scale-down
```

Object-Fit

- fill: O conteúdo é dimensionado para preencher todo o espaço do container. Pode perder sua proporção (aspect ration)
- contain : O conteúdo é dimensionado para manter sua proporção enquanto se encaixa no container.
- cover: O conteúdo é dimensionado para manter sua proporção e preencher toda a caixa de conteúdo. Parte da imagem poderá ser cortada.
- none: O conteúdo não é redimensionado.
- scale-down: O conteúdo é dimensionado como se fosse as opções “none” ou “contain”. A escolha do navegador será aquela que retornar uma imagem menor.

Aspect-Ratio

- Propriedade que define a proporção de um elemento. É a relação entre sua altura e largura. Representada como uma razão entre dois números, por exemplo:
  - aspect-ratio : 16/9

JavaScript

Variáveis e Constantes

Variáveis com *Let* - Definição

- Para criar variáveis em JS utilizamos a palavra-chave "let".
- Dado que JS é uma linguagem de tipagem dinâmica, o tipo só será definido em tempo de execução.

Variáveis com *Let* - Exemplo

```
let x; //declarando a variável cujo identificador é x
x = 3; //atribuindo valor após a definição
console.log(x); // 3
let y = 4; //atribuindo valor no momento da definição
console.log(y);// 4
```

Constantes com *const* - Definição

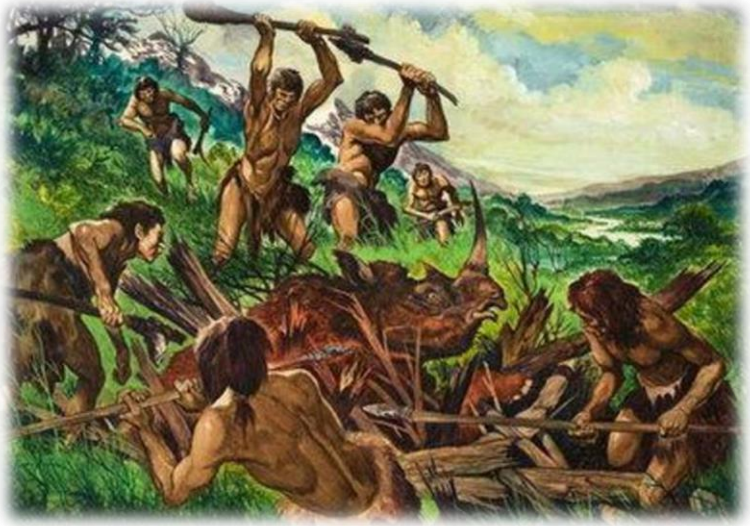
- Área de memória que não pode ser modificada após ter o seu valor atribuído.
- Em JS usamos a palavra-chave "const".
- Uma das principais razões pelas quais é importante usar "const" é para evitar a reatribuição acidental de valores de variáveis e referências.

Constantes com *const* - Exemplo

- O uso de constantes aprimora a legibilidade e ajuda a criar um código mais seguro.

```
const x = 7;
x = 8; //ERRO
```

Tipos Primitivos



Tipos Primitivos em JS

- Number
- Boolean
- String
- Null
- Undefined

Numbers - Definição

- A linguagem JS possui um único tipo de dado para número. Esse tipo é capaz de armazenar números inteiros e valores reais.
- Essa abordagem é diferente de outras linguagens populares como C e Java. Pois estas utilizam tipos diferentes, como “int” e “float”.

Numbers – Exemplos

```
let idade = 34;
console.log(typeof(idade));//number

let telefone = 988998899;
console.log(typeof(telefone));//number

let peso = 84.56;
console.log(typeof(peso));//number

let temp = -14.673561;
console.log(typeof(temp));//number
```

Numbers – Operações Matemáticas

- Podemos executar operações matemáticas básicas com valores do tipo “number” tais como:
  - Soma (+), Subtração (-), Multiplicação (*).
  - Divisão (/), Resto (%), Exponenciação (**).
  - Outros..

Numbers – NaN (Not a Number)

- Algumas operações matemáticas podem resultar em NaN (Not a Number).
- Exemplo: 0/0 irá resultar em NaN
- O valor NaN é do tipo “number” e representa algo que não é um valor numérico.

Numbers – Funções em *Math*

```
let x = 34.56;
let y = -78.12;

//remove a parte fracionária
console.log(Math.floor(x));//34

//arredonda para cima
console.log(Math.ceil(x));//35

//valor absoluto
console.log(Math.abs(y));//78.12

//exponenciação da base 2 pelo expoente 3
console.log(Math.pow(2,3)); //8

//constante PI
console.log(Math.PI); //3.141592653589793
```

Numbers – Incremento/Decremento

```
let x = 6;
console.log(x++);//6
console.log(x);//7
console.log(++x);//8
console.log(--x);//7
```

Numbers – Números Aleatórios

- Geramos números aleatórios usando a função Math.random() que retorna um valor real aleatório entre 0 e 1 (exclusivo).
- É possível obter números inteiros e outras faixas de valores combinando com algumas funções matemáticas.

Numbers – Números Aleatórios - Exemplo

```
//Valor aleatório entre 0 e 9
let x = Math.floor(Math.random() * 10);

console.log(x);
//Valor aleatório entre 1 e 10

x = Math.floor(Math.random() * 10) + 1;
console.log(x);

//Valor aleatório entre 2 e 6
x = Math.floor(Math.random() * 5) + 2;
console.log(x);
```

Boolean - Definição

- Variáveis desse tipo podem ter o valor “true” ou “false”.
- Importante notar que estamos falando do valor true/false e não do literal “true” ou “false”.

Boolean - Exemplo

```
let x = true;
let y = false;
let z = "true"; //diferente de atribuir true ou false sem aspas
console.log(typeof(x)); //Boolean
console.log(typeof(y)); //Boolean
console.log(typeof(z)); //string
```

Boolean - Exemplo

- Como a tipagem em JS é dinâmica, nada impede de atribuirmos um valor de outro tipo a está variável.

```
let x = true;
console.log(typeof(x)); //boolean
x = 1;
console.log(typeof(x)); //number
```

String - Definição

- Representa uma sequência de caracteres. É utilizada para armazenar informações textuais e precisam ser envolvidas por aspas duplas/simples.
- Podem ser envolvidas pelo acento grave (`) para criar *template strings*.

String - Exemplo

```
let nomeUsuario = "mestreDosMagos";
let nome = 'Maria’;
let cidade = `Itajubá`;
let endereco = `Moro em ${cidade}, no estado de Minas Gerais. `;

console.log(typeof(nomeUsuario));//string
console.log(typeof(nome)); //string
console.log(typeof(cidade)); //string
```

String – Comprimento e Concatenação

```
let cidade = 'Itajubá’;
console.log(cidade.length); //7

let nome = "João";
let sobrenome = “ da Silva";
let nomeCompleto = nome + " " + sobrenome;
console.log(nomeCompleto); //João da Silva
```

String – Concatenação com Números

```
let num = 2; //sou number
num += 1;
console.log(num); //3
let literal = "2"; //sou string
literal += 1; //Literal + Number = Literal
console.log(literal) //"21“

literal++ //??????
```

String – Métodos Auxiliares

```
let nome = "   João   ";
let ret;

nome = nome.trim(); //remove espaços em branco no início e fim
ret = nome.toLowerCase(); //todas as letras ficam minúsculas
ret = nome.toUpperCase(); //todas as letras ficam maiúsculas
ret = nome.startsWith("J"); //retorna true se nome se iniciar com "J"
ret = nome.startsWith("o",2); //true se nome se iniciar com "o" no
índice 2
ret = nome.endsWith("o"); //retorna true se nome terminar com "o"
```

String – Métodos Auxiliares com Argumentos

```
let frase = "Eu quero tomar café";

//indice onde se encontra a primeira ocorrência de "E"
frase.indexOf('E’));

//indice onde se encontra a primeira ocorrência de "quero"
frase.indexOf('quero');//3

//sequencia entre os caracteres na posição 0 (inclusivo) e 2 (exclusivo)
frase.slice(0,2); //"Eu“

//sequencia de caracteres a partir da posição 3
frase.slice(3); //"quero tomar café“

//sequencia de 4 caracteres a partir do final da string
frase.slice(-4); //"café“

frase.replace('café','suco');
```

String – Template Literals

- Sequencias de caracteres que permitem interpolação e embutir expressões que serão calculadas e substituídas em tempo de execução em seus *placeholders*.

String – Template Literals Exemplo

```
let precoCafe = 4.50;
let precoCoxinha = 6.00;
const msg = `O café ${precoCafe} e coxinha ${precoCoxinha} resultam no total
de $${precoCafe + precoCoxinha}.`;
console.log(msg);
//O café 4.5 e coxinha 6 resultam no total de $10.5.
```

Null e Undefined – Definição

- O tipo ***null*** se refere a ausência proposital de valor e, portanto, é necessária a operação de atribuição do valor null.
- O tipo ***undefined*** ocorre quando alguma variável não teve nenhum valor atribuído e apenas foi definida.

Null e Undefined – Exemplo

```
let usuarioLogado = null;
console.log(typeof(usuarioLogado)); //null
let x;
console.log(x);//undefined
```

Condicionais

Condicionais – Operadores de Comparação

- Podemos comparar valores e verificar se resulta em um valor verdadeiro ou falso

>	**maior que**
<	menor que
>=	maior ou igual
<=	menor ou igual
==	igual
!=	diferente
===	estritamente igual
!==	estritamente diferente

Condicionais – Operadores de Comparação

- Diferença entre a igualdade (==) e estritamente igual (===)

```
5 == '5'; //true
5 === '5'; //false
0 == false; // true
0 === false; //false
undefined == null; //true
undefined === null; //false
```

Condicionais – Estrutura If-Else

- Estrutura tradicional presente em diversas linguagens para testar condições.

```
if(condição){
    //executa se a condição é verdadeira
}else{
    //caso contrário
}
```

Condicionais – Estrutura If-Else

- Encadear diversos *else-if* com novas condições

```
if(condição1){
    //executa se a condição1 é verdadeira
}else if(condição2){
    //executa se a condição2 é verdadeira
}else if(condição3){
    //executa se a condição2 é verdadeira
}else{
    //executa se todas as condições forem falsas
}
```

Condicionais – Operadores Lógicos

- Operadores binários cujos operandos são expressões lógicas.
- O resultado é um valor booleano

AND (E)	**&&**
OR (OU)	**||**
NOT (NEGAÇÃO)	**!**

Condicionais – Valores Booleanos Internos

- Todo valor em JS, isoladamente, retorna um valor verdadeiro com exceção dos seguintes:
  - false
  - 0
  - “” (string vazia)
  - Null, undefined e NaN

Condicionais – Estrutura Switch

```
switch (valor testado) {
    case valor1:
        break;
    case valor2:
        break;
    default:
        break;
}
```

Arrays

Arrays – Exemplo Inicialização

```
//array vazio
let discentes = []

//array de strings
let cores = ['red','green','blue’];

//array de numbers
let versoesWindows = [95,98,7,8,8.1,10,11];

//array com tipos diferentes
let coisas = [true, 18,'café',null];


console.log(cores[1]);//green

console.log(versoesWindows[3]);//8

console.log(versoesWindows.length);//7
```

Arrays – Exemplo Atualizando Valores

```
let cores = ['red','green','blue’];
console.log(cores[1]);//green
cores[0] = 'yellow'; //Modificando o valor da posição 0 para yellow
console.log(cores[0]);//green
```

Arrays – Função Push/Pop

```
let num = [1,2,3];
console.log(num.length); // 3
num.push(4);
num.push(5,6);
console.log(num); //[1, 2, 3, 4, 5, 6]
console.log(num.length);//6
num.pop();
console.log(num); //[1, 2, 3, 4, 5]
console.log(num.length);//5
```

Arrays – Função Shift/Unshift

```
let num = [1,2,3];
console.log(num.length); // 3
num.unshift(4);
num.unshift(5,6);
console.log(num); //[5, 6, 4, 1, 2, 3]
console.log(num.length);//6
num.shift();
console.log(num); //[6, 4, 1, 2, 3]
console.log(num.length);//5
```

Arrays – Função Concat/Includes/IndexOf

```
const arr1 = ['a','b','c’];
const arr2 = ['d','e','f’];
const arr3 = arr1.concat(arr2);
console.log(arr3);//['a', 'b', 'c', 'd', 'e', 'f']
//includes verifica se um valor está presente
console.log(arr3.includes('b'));//true
console.log(arr3.includes('g'));//false
//indexOf verifica a posição de um valor presente.
console.log(arr3.indexOf('c'));//2
//Caso o valor não esteja presente, a função retorna -1
console.log(arr3.indexOf('g'));//-1
```

Arrays – Função Reverse

```
const arr1 = ['a','b','c’];
const arr2 = ['d','e','f’];
const arr3 = arr1.concat(arr2);
console.log(arr3);//['a', 'b', 'c', 'd', 'e', 'f']
//Reverse irá sobrescrever o array com os valores reversos
arr3.reverse();
console.log(arr3);//['f', 'e', 'd', 'c', 'b', 'a']
```

Arrays – Função Slice

```
const arr1 = ['a','b','c’];
const arr2 = ['d','e','f’];
const arr3 = arr1.concat(arr2);
console.log(arr3);//['a', 'b', 'c', 'd', 'e', 'f']
//Slice irá retornar um cópia de uma porção do array
//de acordo com os índices passados.
const arr4 = arr3.slice(1);
console.log(arr4);//['b', 'c', 'd', 'e', 'f']

const arr5 = arr3.slice(2,4);
console.log(arr5);//[‘c', ‘d']
```

Repetição

Repetição – Definição

- Estrutura de controle que executa um trecho de código ***enquanto*** uma dada condição for verdadeira.
- JS conta com as tradicionais estruturas *for* e *while*.

Repetição – *For*

- ***For*** tradicional utilizando um contador

```
let num = [1,2,3,4,5]

for(let i = 0; i < num.length; i++)
    console.log(num[i]);
```

Repetição – *For - in*

- ***For*** utilizado para percorrer todas as propriedades de um objeto.

```
const carro = {
    nome: "Jaum",
    idade: 32,
  };

for(const i in carro)
    console.log(i, carro[i]);
```

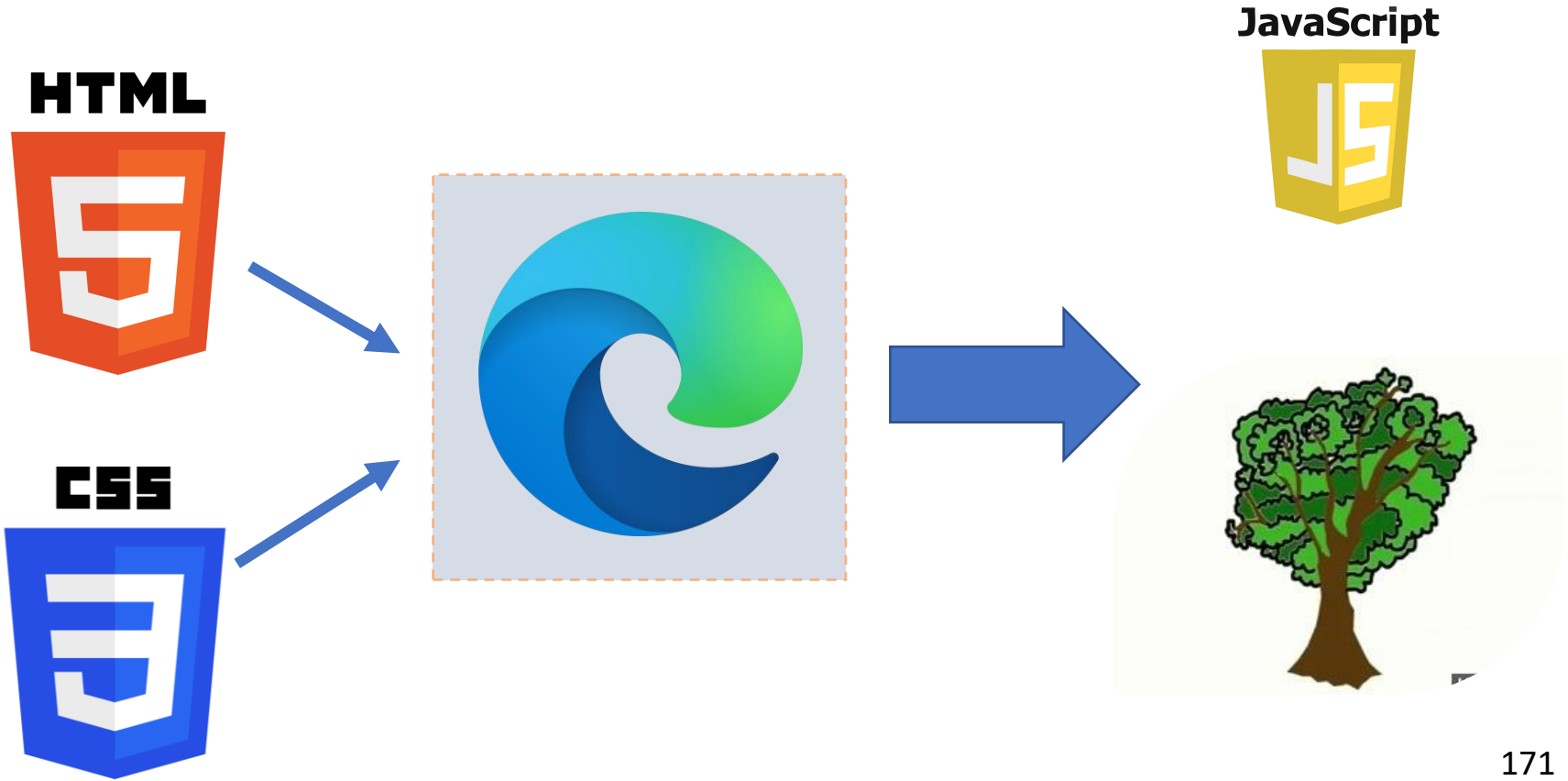
Repetição – *For - of*

- ***For*** utilizado para percorrer coleções

```
let num = [1,2,3];

for(const valores of num)
    console.log(valores);
```

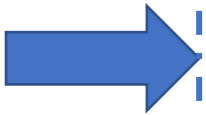
DOM

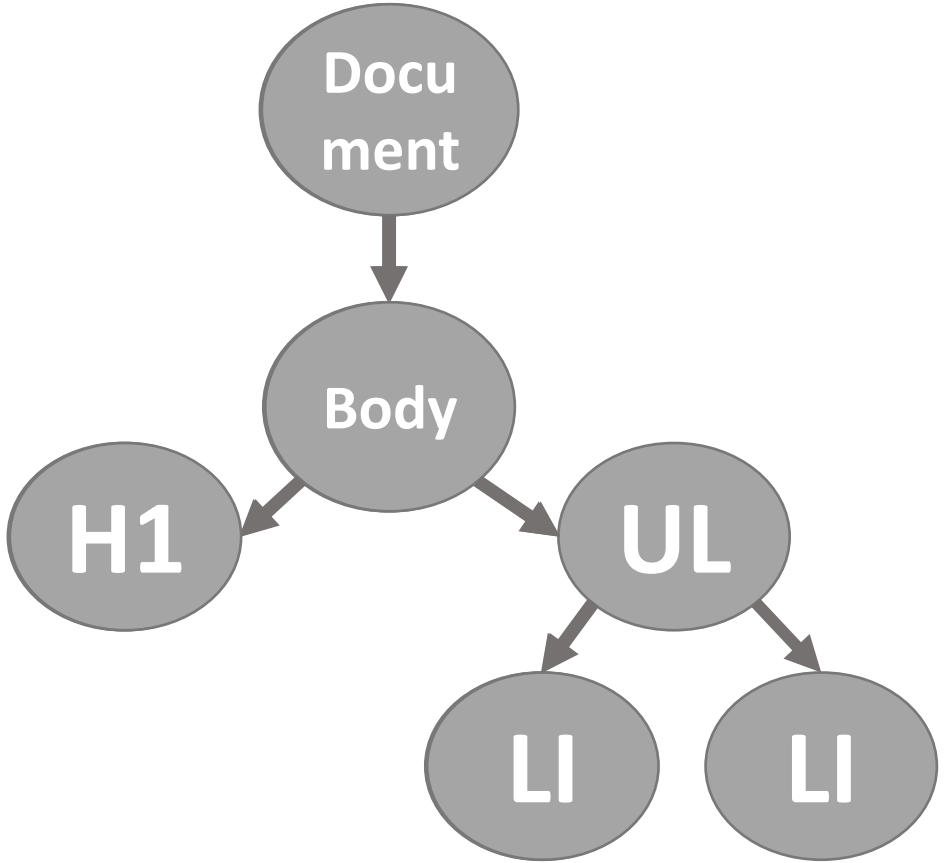
Navegador Cria o DOM



DOM - Árvore

```
<body>
    <h1>Tarefas da semana</h1>
    <ul>
        <li>Estudar Web</li>
        <li>Estudar C++</li>
    </ul>
</body>
```





Visualizar o DOM

- É possível invocar o objeto DOM diretamente no JS com *document*.
- O comando console.dir(document) apresenta toda a estrutura da página em formato objeto Javascript

Seleção de Elementos

querySelector - Definição

- Com o avanço do JS, um novo método foi adicionado ao DOM.
- Este permite fazer a seleção da mesma forma que é feita no CSS.
- É possível buscar por id, class, atributos, e todas as outras combinações usadas no CSS.

querySelector - Exemplos

- document.querySelector('#id’);
- document.querySelector(‘.class’);
- document.querySelector(‘nome-tag’);
- Em todos esses casos, o retorno é apenas um elemento. Para retornar uma lista, se usa o método querySelectorAll().

querySelector – Exemplos ID

- Retornando elemento H1 por ID.

```
const h = document.querySelector("#heading");
console.log(h);
```

querySelector – Exemplos Class

- Retornando o primeiro botão com a classe especificada

```
const btn = document.querySelector(".calculadora-tecla-operador");
```

querySelector – Exemplos Tag

- Retornando o primeiro botão buscando pela *tag*.

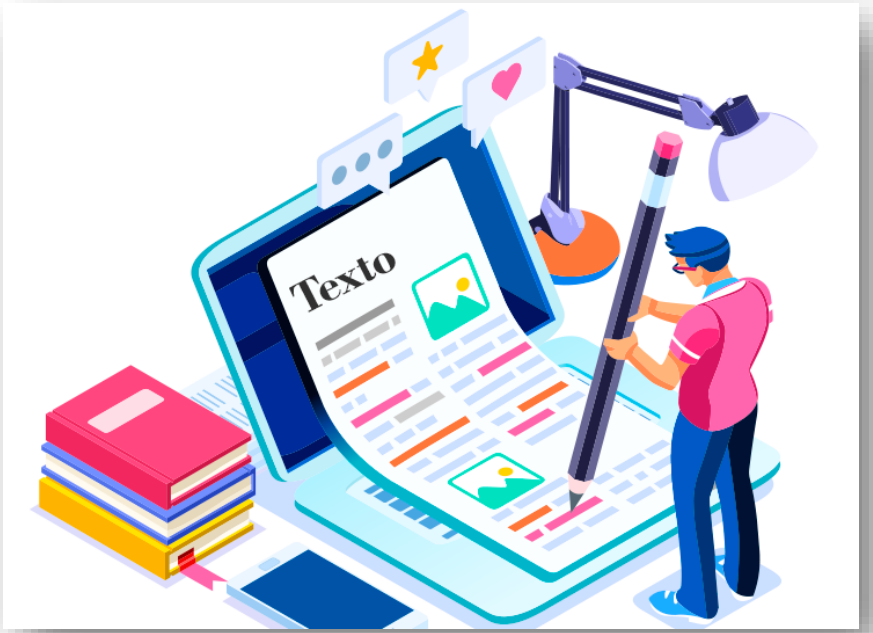
```
const elemento = document.querySelector("button");
```

querySelectorAll – Exemplos

- Retornando todos os botões com a classe especificada.

```
const btnS = document.querySelectorAll('.calculadora-tecla-operador');
console.log(btnS.length);//4
```

Conteúdo Textual



Conteúdo Textual

- Podemos manipular o conteúdo textual dentro dos elementos HTML com as seguintes propriedades:
  - innerHTML
  - innerText
  - textContent

Conteúdo Textual - innerText

- Retorna o texto visível que se encontra no elemento HTML.
- Se algum texto estiver com a condição ‘hidden” não será mostrado

Conteúdo Textual - textContent

- Retorna o texto completo que se encontra no elemento HTML, mesmo que esteja “Hidden”.

Conteúdo Textual – Exemplo innerText e textContent

- HTML Exemplo:

```
<p id="paragrafo">
  Eu sou um paragrafo.
  <span style="display: none;">Eu estou
hidden.</span>
</p>
```

- JavaScript:

```
//Eu sou um paragrafo.
document.querySelector('#paragrafo').innerText;

//Eu sou um paragrafo. Eu estou hidden.
document.querySelector('#paragrafo').textContent
```

Conteúdo Textual - innerHTML

- Retorna o texto completo que se encontra no elemento HTML, incluindo HTML interno, isto é, elementos HTML descendentes.

Conteúdo Textual – innerHTML Exemplo

```
<p id="paragrafo">
  Eu sou um paragrafo.
  <span style="display: none;">Eu estou
hidden.</span>
</p>
```

```
//Eu sou um paragrafo. <span style="display: none;">Eu estou hidden.</span>
document.querySelector('#paragrafo').innerHTML;
```

Conteúdo Textual - Modificações

- Com essas propriedades é possível também alterar o conteúdo e com isso sobrescrever utilizando o operador de atribuição.

```
document.querySelector('#paragrafo').innerText = 'Novo Texto';
document.querySelector('#paragrafo').innerText += ' Concatenando';
```

Acessando os Atributos

Acessando Atributos

- Os atributos dos elementos HTML podem ser acessando como propriedades do objeto JS. Pode-se utilizar o operador ponto (.) para acesso.
- Esse cenário é interessante quando se está criando os elementos, e deseja preencher os valores dos atributos.

Acessando Atributos - Métodos

- É possível também utilizar dois métodos para acessar e modificar os atributos.
- getAttribute() para acessar
- setAttribute() para modificar

Acessando Atributos - Exemplo

- Modificar a descrição da imagem, isto é, o atributo 'alt'. Usando as duas formas.

```
const mapaMG = document.querySelector('#minas-gerais img’);

mapaMG.alt = 'O mapa de minas gerais';

mapaMG.setAttribute('alt', 'O mapa de minas gerais');
```

Estilizando via JavaScript

Modificando as propriedades do CSS - inline

- As propriedades de estilização podem ser modificadas diretamente no objeto JS, mas estas estão escritas em formato *Camel Case*.
- As propriedades inseridas via JS serão *inline* e pode não ser conveniente adicionar nesse formato.
- A propriedade *style* é acessada primeiro.

Modificando as propriedades do CSS - Exemplo

- Cada propriedade é acessada separadamente

```
const pp = document.querySelector('#paragrafo');
pp.style.fontSize = '5em';
pp.style.color = 'blue';
```

Acessa as propriedades do CSS

- As estilizações que não são inline, não conseguem ser lidas diretamente pelo propriedade *style*
- É preciso acessar o objeto global *window* e o método *getComputedStlye()* passando como parâmetro o objeto JS.
- Com isso é possível acessar as propriedades.

Acessa as propriedades do CSS - Exemplo

```
const pp = document.querySelector('#paragrafo');
pp.style.color = 'red’;

window.getComputedStyle(pp).color;//rgb(255, 0, 0)

window.getComputedStyle(pp).fontSize;// 16px
```

Modificando com *classList*

- Uma forma mais interessante de estilizar a partir do JS é manipulando as classes CSS.
- Para isso usamos a propriedade *classList*
- Esta apresenta uma API que permite adicionar, remover e ler as classes utilizadas no elemento HTML.

Modificando com *classList*

- classList.add()
- classList.remove()
- classList.toggle() // desliga/liga a classe

```
const pp = document.querySelector('#paragrafo');
pp.classList.add('caixa-p');
```

Percorrendo a Árvore DOM

- parentNode
  - Propriedade que devolve o elemento pai.
- childElementCount
  - Retona o número de descendentes.
- children (HTMLCollection)
  - Retorna uma coleção com os descentes.

Adicionando Elementos

- createElement(“nome tag”)
- append(“elemento ou texto”)
- appendChild(“apenas elementos”)
- prepend(“elemento ou texto”) //add no início
- after()//Adiciona após

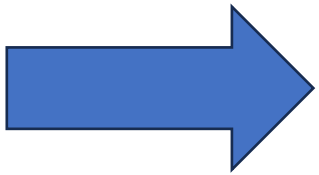
Adicionando Elementos - Adjacente

- inserAdjacentElement(“local” , elemento)
  - Local pode ser:
    - afterbegin => após o início
    - Afterend => após o fim
    - Beforebegin => antes do início
    - beforeend => antes do fim

Adicionando Elementos - Exemplo

- Adicionar dois <li> após início do

```
//Já existe um ul na página
const ul = document.querySelector('ul');

const li1 = document.createElement('li');
li1.innerText = `Item 1`;
const li2 = document.createElement('li');
li2.innerText = `Item 2`;

//Inserir li1 após inicio
ul.insertAdjacentElement('afterbegin', li1);
//Inserir li2 após inicio
ul.insertAdjacentElement('afterbegin',li2);
```



- Item 2
- Item 1

Removendo Elementos

- Element.remove();

```
const img = document.querySelector('img');
img.remove();
//Removeu a imagem do DOM
```

DOM
Eventos

Evento de Click

- É necessário adicionar um “listener” para o evento ‘click’.
- Se for um <button> certificar que o tipo é “button” e não “submit”

Evento de Click - Exemplo

```
const btn = document.querySelector('button');

const funcao = (e) => {
    //A variável "e" é uma referência para o evento.
    //É possível navegar no elemento (target) que disparou
    //e manipular suas propriedades
    console.log('Botão clicado');
}

btn.addEventListener('click', funcao);
```

Evento de Change

- Pode ser utilizado para detectar que mudou um valor em um <select>

Evento de Change - Exemplo

```
const select = document.querySelector('select');

//Valor inicial do select
let valorInicial = select.value;

const mudou = (e) => {

    console.log(e.target.value); //novo valor

}

select.addEventListener('change', mudou);
```