

Fundamentos Front-End  
HTML5, CSS3 e JAVASCRIPT

# Anotações do Curso

Aluno: João Vitor Villas Bôas

# Parte 1 – HTML e CSS (Folhas de Estilo)

## 1.Declaração e Referências.....1

1.1 Declaração do CSS	1
1.2 Referência: Onde Escrever as Folhas de Estilo	1
1.3 Referência Através de Classes	2
1.4 Referência Através de ID	2

## 2.Algumas Propriedades Úteis, Funções e Unidades de Medida no CSS.....2

2.1 Algumas Propriedades	2
2.2 Algumas Funções	3
2.3 Unidades de Medida	3

## 3.Atalho Para Digitação HTML: EMMET no VISUAL STUDIO CODE.....3

## 4.Principais Elementos HTML e Propriedades CSS.....4

4.1 Body	4
4.2 Outras Tags	4
4.3 Propriedades de Fontes	5
4.4 Propriedade de Texto	5
4.5 Propriedades de Cor e Cor de Fundo: Color e Background-Color	6
4.6 Box Model	7
4.7 Pseudo Classes em Links	8
4.8 List Style ( Formatando os Bullets das Listagens)	9
4.9 Trabalhando com Imagens e Background – Plano de Fundo	9
4.10 Estrutura do Layout (Wireframe)	10
4.11 Propriedade Float e Clear	12
4.12 Propriedade Display	13
4.13 Propriedade White-Space	13
4.14 Dicas Para Construir um Layout (Exemplo)	
• Fazer menu Horizontal	14
• Construção cards laterais	
4.15 Propriedade Position	17
4.16 Propriedades Z-Index	19
4.17 Listas	19
• Fazendo aparecer uma sub-lista, quando passar o mouse sobre um item da lista	
• Fazendo uma animação de lista, como se fosse um menu com submenu	
4.18 Propriedade Overflow	21
4.19 Tabelas	21
4.20 Formulários	23
4.21 Efeitos Interessantes com CSS	26
• Border-radius	
• Box-shadow	
• Text-shadow	
• Gradiente Lineares	
• Gradiente radial	

<ul style="list-style-type: none"> <li>• Gradiente repetitivos</li> <li>• Fontes customizadas</li> </ul>	
4.22 Transições Suaves	29
4.23 Dicas no Projeto Strata	31
<ul style="list-style-type: none"> <li>• Construindo o menu lateral fixo, utilizada com &lt;HEADER&gt;</li> <li>• Colocar elementos em duas colunas</li> <li>• Alinhamento do formulário Get in Touch</li> <li>• Colocar um elemento invisível</li> </ul>	
4.24 Seletores Compostos	35
<ul style="list-style-type: none"> <li>• Filho Direito</li> <li>• Irmão Adjacente</li> <li>• Próximo Irmãos</li> </ul>	
4.25 Seletores de Atributos	36
<ul style="list-style-type: none"> <li>• Atributo qualquer</li> <li>• Atributo Específico Exato</li> <li>• Atributo Específico com Valor Inicial</li> <li>• Atributo Específico com Valor Final</li> <li>• Seletor de negação</li> </ul>	
4.26 Incluir Conteúdo Dinamicamente com CSS: Content, After e Before	37
4.27 Pseudo-Classes e Pseudo-Elementos	39
<ul style="list-style-type: none"> <li>• Focus</li> <li>• First-child</li> <li>• Last-child</li> <li>• nth-child (odd ou even)</li> <li>• nth-of-type (N)</li> </ul>	
4.28 Propriedade Transform 2D	41
4.29 Criando um CheckBox Estilizado	42

## **5.RWD-Responsive Web Design (media queries).....43**

## **6.Imagens Responsivas.....45**

6.1 Tags Figure e FIGCAPTION	46
6.2 Tag <PICTURE>	46

## **7.Incorporação de Conteúdo (Áudio, Vídeo e Iframe).....47**

## **8.Flexbox (ou Display Flex).....47**

- Fazendo os cards com mais facilidade
- Alinhando os elementos de cada card

## **9.Grid layout (ou Display Grid ou CSS Grid).....51**

- Definindo o Grid
- Mudando a posição dos elementos na Flex Grid
- Mesclagem de elementos (células)
- Criando Template Areas
- Gridlines nomeadas
- Auto-fill vs auto-fit e auto-rows

## 10.Mais sobre Semântica, Especificidade e Dica de Altura nos Elementos...56

## 11.Animação.....57

### Parte II – Javascript

## 12.Introdução ao Javascript.....61

- Depuração de Código
- Integração HTML e Javascript

## 13.Algumas Funções, Objetos, Propriedades e Métodos Importantes.....62

## 14.Sintaxe e Lógica de Programação em Javascript.....63

14.1 Variáveis	63
14.2 Constantes	63
14.3 Conversão de Dados	63
14.4 Operadores	64
• Atribuição	
• Concatenação	
• Conversão em booleano	
• Aritméticos	
• Comparação	
• Lógicas	
14.5 Estruturas de Controle	65
• Condicionais	
• Operador ternário	
• Repetições	
14.6 Funções	66
• Declarações, chamadas e argumentos	
• Arrow Function	
14.7 Tratamentos de Erros	70
14.8 Arrays	70

## 15.Objetos em Javascript.....71

- Definição
- Arrays de Objetos

## 16. DOM (Document Object Model).....73

16.1 Atributos e Métodos do DOM (Object Document)	73
• Atributos e Métodos das instâncias do DOM	
• Capturando o valor de um <input type="text">	
• Trabalhando com os elementos filhos	

## 17.Objetos em Javascript.....79

## 18.Eventos em Javascript.....80

18.1 Sintaxe (DOM nível 0)	80
18.2 Sintaxe (DOM nível 2)	81
18.3 Alguns Eventos Comuns ( DOM 2)	82

18.4 Propagação de Eventos (bubbles)	82
18.5 O Objeto Event	83
18.6 Delegação de Eventos	84
18.7 Validação de Formulários, com o botão Submit	84
18.8 Atribuindo Eventos a Nodelist	85

## **19.String em Javascript.....86**

19.1 Declaração	86
19.2 Métodos e Propriedades	86

## **20.Números em Javascript (Objeto Number).....87**

20.1 Declaração	87
20.2 Métodos e Propriedades	87

## **21.Objetos Math.....88**

## **22.Arrays em Javascript.....89**

22.1 Propriedades e Métodos	89
22.2 Spread Operator	92

## **23.Objeto Date.....92**

## **24.Intervalos(Temporizadores).....93**

## **25.BOM (Browser Object Model).....94**

25.1 Objeto Window	94
25.2 Objeto History	94
25.3 Objeto Location	94
25.4 Objeto Navigator	95
25.5 Objeto Screen	95
25.6 BOM: Métodos	95
25.7 BOM: Eventos	95
25.8 Estilização (getComputedStyle())	96
25.9 Acessando Posições e tamanhos em relação ao Document	96
25.10 Acessando Posições e tamanhos em relação ao Viewport	96

## **26.Controle de Fluxo: Return, Break e Continue.....96**

## **27.Conceito Progressive Enhancement.....97**

## **28.Projeto Final – Anotações.....97**

28.1 Ferramentas de Desenvolvimento e Bibliotecas de Suporte	97
28.2 Técnicas	97
<ul style="list-style-type: none"> <li>• Para Saber se o Javascript está ativo</li> <li>• Desenhar seta na parte inferior da capa</li> </ul>	

## **29. Servidores para Publicação e Registro de Domínio.....98**

**Tabela de Recurso e Ferramentas Sugeridas:**

<b>Editores de Código</b>	<b>Visual Studio Code</b>	<b><a href="https://code.visualstudio.com/">https://code.visualstudio.com/</a></b>
<b>Formatação e padronização</b>	<b>Resetando os padrões de pré-formatarão dos browsers (Salvar como arquivo CSS e linkar ao arquivo HTML – recomendável que seja o PRIMEIRO CSS a ser vinculado).</b>	<b><a href="https://necolas.github.io/normalize.css/">https://necolas.github.io/normalize.css/</a></b>
<b>Compatibilidade</b>	<b>Verificar Suporte dos browser aos elementos</b>	<b><a href="https://caniuse.com/">https://caniuse.com/</a></b>
<b>Interface</b>	<b>Fontes e ícones</b>	<b><a href="https://www.fontsquirrel.com/">https://www.fontsquirrel.com/</a> <a href="https://fonts.google.com/">https://fonts.google.com/</a> <a href="https://fontawesome.com/">https://fontawesome.com/</a> <a href="https://icomoon.io/">https://icomoon.io/</a></b>
<b>Transições</b>	<b>Definição de movimento nas transições</b>	<b><a href="https://cubic-bezier.com">https://cubic-bezier.com</a></b>
<b>Hospedagens e Registro</b>		<b><a href="http://www.netlify.com">www.netlify.com</a> <a href="http://registro.com/">http://registro.com/</a> <a href="https://br.godaddy.com/">https://br.godaddy.com/</a></b>

## Parte I – CSS (Folhas de Estilo)

### 1. Declaração e Referências:

#### 1.1 Declaração das Folhas de Estilo (Css)

Seletor {propriedade: valor;}

**Seletor:** o elemento que se pretende formatar (através da tag HTML, do ID, da classe).

**Exemplos:**

```
H1 {color: red;}

H1 {
    Color: red;
    Font-size: 16px;
}
```

Pode-se utilizar a mesma declaração para mais de um seletor. Exemplo:

```
p, h1, li {
    ...
}
```

Pode-se definir seletores específicos de uma classe

```
.hero-content h1 { //formata apenas o <h1> que estiver dentro da classe .hero-content
    ...
}
```

#### 1.2 Referência: onde Escrever as Folhas de Estilos

- **INLINE:** Dentro da própria tag no arquivo HTML. Exemplo: <p style="...">...</p>

- **INTERNO:** Entre as marcações <style>...</style>, dentro de <HEAD></HEAD> do arquivo HTML.

- **ARQUIVO EXTERNO:** Escrever a folha de estilo em um arquivo separado, com extensão .css, e vinculá-lo ao HTML através da instrução <link href="local\_do\_arquivo.css" rel="stylesheet">.

**1.3 Referência Através de Classes:** Aplicável quando se deseja formatar apenas os elementos que possuem a mesma classe.

- Declaração da classe (CSS):  
 .minha-classe{...}  
 .classe-pai classe-filha {...}  
 .classe1 + classe2 {...} //afeta todos elementos classe2 que vier imediatamente após classe1.
- Referência no HTML:  
 <p class="minha-classe">...</p>

Pode-se aplicar mais de uma classe ao mesmo elemento , da seguinte forma:

<p class="classe1 classe2">....</p>

**1.4 Referência através de ID:** Aplicável quando apenas um único elemento terá propriedades específicas. Os ID's são únicos, ou seja , não se pode ter o mesmo ID para vários elementos.

- Declaração do ID (CSS):  
 #meu-id { ... }
- Referência (HTML):  
 <p id="meu-id">...</p>

## 2. Algumas Propriedades Úteis, Funções e Unidades de Medidas no CSS:

### 2.1 Algumas Propriedades:

Cursor: pointer; // ponteiro do mouse com cursor de link

Background-clip: border-box; (padrão) => A borda está incluída no conteúdo da <DIV>  
 Content-box: limites do conteúdo da <DIV>  
 Padding-box: conteúdo da <DIV> não inclui borda

tabindex = "x"; // atribui a ordem "x" para elementos. Força receber TAB.



## 2.2 Algumas funções:

Calc(...); // retorna o resultado de um cálculo.

## 2.3 Unidades de Medida:

**px**: valor absoluto em pixels.

**%**: valor relativo, percentual em relação ao elemento container.

**em**: valor relativo, em relação ao tamanho da fonte do elemento container.  
Por padrão, o tamanho da fonte é 16px.

**rem**: valor relativo, em relação ao tamanho da fonte do elemento principal (root - HTML).  
Por padrão, o tamanho da fonte é 16px.

## 3. Atalhos Para Digitação HTML: Emmet Visual Studio:

<elemento>[<atributo>]{<texto>}><elementos internos>+<elemento lateral>\*<qtd>

Símbolos EMMET e sua representação no VS Code:

Símbolo	Ação
[ ]	Define um atributo
.	Define uma classe
#	Define um ID
{...}	Define um texto para o elemento
+	Cria um elemento "irmão"
>	Cria um elemento "filho"
*	Define quantas vezes esse elemento será criado

Exemplo 1:

```
div#minha_div{Minha Div}>h1{Sub Título}>p*3
```

Resultará em:

```
<div id="minhdiv">
  Texto da Div
  <h1>
  Título
  <p></p>
  <p></p>
  <p></p>
</h1>
</div>
```

Exemplo 2:

```
input[type=text].txtNome+button#meuBotao{Clique Aqui!}
```

Resultará em:

```
<input type="text" class="txtNome"><button id="meuBotao">Clique Aqui!</button>
```

## 4. Principais Elementos HTML e Propriedades no CSS:

### 4.1 Body:

**Width:** Definir a largura do documento, em relação à tela. Se for definido como auto, voltará à largura padrão, desfazendo eventual alteração.

**Max-width:** Tamanho máximo que pode chegar um elemento, em relação a si próprio. um maxwidth: 100% em uma body img{ significa que em nenhuma hipótese a imagem vai ficar maior que seu próprio tamanho.

**Margin:** define as margens. Para centralizar todo o conteúdo da página em relação à tela, basta informar a margem esquerda, que será duplicada para a margem direita. Atentar para as proporções entre width e margin.

Exemplo:

```
body {
  width: 60%; /*sobrou 40% da tela*/
  margin-left: 20%; /*20% de margem esquerda, sobrando
  20% para a margem direita.*/
}
```

O atributo **margin:auto**; separa automaticamente o espaço que ficou sobrando através do width, para os lados esquerdo e direito (centraliza de forma automática). O auto não funciona para o alinhamento vertical.

Exemplo:

Margin {30px, auto} //atribui margem superior e inferior de 30px, e alinha horizontalmente automaticamente.

### 4.2 Outras Tags:

- <sup>...</sup>: sobrescrito
- <sub>...</sub>: subscrito
- <b>...</b>, <i>...</i>, <u>...</u>: negrito, itálico e sublinhado.
- <code>...</code>: representa/formata/exibe como trecho de código.
- <blockquote cite="URL"><p>Esta é uma citação extraída de URL</p></blockquote>: Utilizado para citações longas.

- **<q cite="URL"><p>**Está é uma citação extraída de URL**</p></q>**: Utilizado para citações curtas.
- **<pre>...</pre>**: Exibe o texto do mesmo jeito que foi digitado, respeitando os espaços, quebras de linha e etc. Útil se quiser exibir trechos de código, respeitando a tabulação digitada no HTML cru, por exemplo. Pode-se aninhar com **<code></code>**.
- **<hX>...</hX>**: Títulos, onde X varia de 1 (maior fonte) a 6 (menor fonte).

Exemplo:

```
.h1, h2, h3, h4, h5, h6 {
  Margin: 0, 0, 1em, 0;
}
```

Atribui uma margem inferior de 1em para todos os títulos <h1> a <h6>.

A unidade em indica que a margem será reduzida proporcionalmente ao tamanho na fonte (font-size).

Se definir 2em, a margem terá duas vezes a proporção da fonte.

#### 4.3 Propriedades de Fontes:

**Font-family:** Tipo de fonte.

**font-family:** Verdana, Geneva, Tahoma, sans-serif;

**Font-size:** tamanho da fonte.

**Font-style:** normal ou itálico.

**Font-weidth:** normal ou negrito.

**Font-variant**

**Color:** define a cor do texto.

#### 4.4 Propriedades de Texto:

**Text-align** (alinhamento)

**Text-decoration** (sublinhado ou não)

**Text-indent** (espaçamento no começo do parágrafo)

**Text-transform** (tudo em maiúsculo, ou minúsculo, ou não)

**Letter-spacing:** espaçamento entre caracteres.

**Line-height:** espaçamento entre linhas. Exemplo:

P {

Line-height: 1.5; //espaçamento entrelinhas de 1,5.

}

Text-shadow. Requer quatro parâmetros: deslocamento horizontal, vertical, intensidade blur e cor.

Exemplo:

Text-shadow: 3px 3px 5px red;

#### 4.5 PROPRIEDADES DE COR E COR DE FUNDO: COLOR E BACKGROUND-COLOR

**Background-color:** cor do fundo de um elemento.

**Color:** cor da fonte de um elemento.

A cor pode ser informada pelos seguintes padrões:

- Padrão RGB em hexadecimal, com o sinal #: #FFFFFF. Quando forem todos os pares repetidos, pode-se abreviar para #FFF.
- Função rgb(R, G, B), onde R, G e B são números inteiros entre 0 e 255.
- Função rgba(R, G, B, A), onde A (alfa) é o nível de TRANSPARÊNCIA, com valores entre 0 (totalmente transparente) e 1 (totalmente opaco).
- inherit: cor herdada da classe pai.

- HSL/HSLA:

Mais fácil para trabalhar com variações de um mesmo tom.

hsl(TOMdeg, SATURAÇÃO%, LUMINOSIDADE%);

TOM: 0deg = vermelho

SATURAÇÃO: 100% = cor mais pura, 0% = tende ao cinza.

LUMINOSIDADE: 0% = preto, 100% = branco, 50% = cor pura definida no TOM.

HSLA: Atribui transparência, da mesma maneira do RGBA.

## 4.6 BOX MODEL:

Margin-<lado>

Padding-<lado>

Width

Height

Border. Três definições: border-style, border-width, border-color.

Visibility: hidden; //Esconde o elemento, mas mantém o espaço reservado em tela.

A propriedade **outline** representa a borda por cima do elemento.

Já **outline-offset** define o distanciamento da borda outline do elemento, que pode ser em valores positivos (mais externa) ou negativos (mais interna).

Sintaxe abreviada (margin ou padding):

Exemplo: p {

margin: 50px 40px 30px 20px; /\* top, right, bottom, left \*/

}

Se omitir um desses valores, o browser vai duplicar o oposto

(right duplica para o left/ topduplica para bottom).

No caso, se omitir um único valor vai sempre duplicar o right. Se colocar apenas um valor, vai atribuir para todos os lados.

Sintaxe abreviada (border):

**H1** {

border: dotted 5px red);

}

O tamanho do elemento será o somatório da propriedade width (se largura) ou height (se altura), somado ao padding. Por exemplo: se tem 100px de largura e 200px de altura, com padding de 25px, a largura passará a ser 150px e a altura 250px. Para fazer com que o padding não seja somado ao heigth/width, de forma que o elemento tenham os valores definidos nessas propriedades, pode-se utilizar o atributo **box-sizing: borderbox;**

#### 4.7 PSEUDO CLASSES EM LINKS

[a:link { ... }](#) //estado padrão  
[a:visited { ... }](#) //link já visitado  
[a:hover { ... }](#) //pode-se utilizar também em outros elementos, como <img>  
[a:active { ... }](#) //no momento do clique

Se quiser aplicar efeitos para todos os estados, utiliza-se apenas o seletor **a**.

O local onde a página será aberta pelo link é definido pelo atributo **target** dentro de **<a>**, indicando se o link será aberto na mesma janela, em aba diversa e etc.

Para alterar os eventos de alguns links e de outros não, utiliza-se a classe. Por exemplo:

```

.menu-main{
  display:inline-block;
  float:right;
  margin-top: 0px;
  padding: 0px;
  margin-right: 30px;
}
.menu-main ul li a{
  font-size: 12px;
  display: inline-block;
  text-transform: uppercase;
  text-decoration: none;
14  color:#1c4559;
  letter-spacing: 2px;
  margin-right: 20px;
}
.menu-main ul li a:hover{
  color: rgba(255, 255, 255, 0.5);
}
/* altera apenas os hover dentro da <ul><li> da .menu-main */
  
```

Propriedade **transition**, dentro de **a** (detalhes no capítulo TRANSIÇÕES SUAVES):

transition: <propriedade que deseja mudar> <tempo c/ unidade><tipo de transição>;

Exemplo:

```

a{
  transition: color 0.3s linear;          /*linear: sem aceleração nem desaceleração*/
}
  
```

#### 4.8 LIST-STYLE (FORMATANDO OS BULLETS DAS LISTAGENS):

List-style-type: //tipo da lista (formato)

List-style-position: inside ou outside //dentro ou fora da lista

List-style-image

Existem tipos específicos para <li> ordenadas (membro de <ol>) ou desordenadas (membro de <ul>).

Declaração:

```
ul li { //Formata toda <li> que estiver dentro de <ul> apenas.
...
}
```

#### 4.9 TRABALHANDO COM IMAGENS E BACKGROUND – PLANO DE FUNDO (AULA 24)

HTML: 

CSS: background-image: url("endereço");

Atributos CSS:

Float: faz a imagem flutuar em relação ao texto (alinhar).

Background-image: Adiciona imagem como background.

Background-repeat: no-repeat; //Evita que o background seja repetido ao longo de todo o objeto. Existe também o repeat-x e o repeat-y como valor para background-repeat.

Background-attachment:

Scroll: background duplicado até o final do objeto, e gera um scroll em caso do fundo ser maior que o elemento.

Fixed: background não repete (equivalente ao background-repeat:no-repeat) e se mantém fixo dentro do documento.

Background-size: tamanho da imagem de fundo dentro do objeto.

Exemplo:

```
Body { background-size: 400px 300px } //400px de largura e 300px de altura.
```

//se Informar apenas 1 atributo, o outro é

definido pela proporcionalidade da imagem

//Pode-se utilizar porcentagem

Valores pré-definidos para o atributo background-size, que podem ser utilizados em substituição aos valores absolutos como mostrado acima:

contain: o background vai preencher todo o objeto, ainda que se repita.

cover: o background vai ser expandido, de forma a caber uma única vez no objeto. Se a imagem for maior que o objeto, a parte excedente é cortada.

auto: mantém o tamanho real da imagem.

Uma imagem dentro de uma <DIV>, que não seja background, pode ser posicionada de forma semelhante ao background-size. No entanto, será utilizada a propriedade **object-fit**.

Background-position: Posição da imagem de fundo, em relação ao objeto que o contém.

Exemplo:

P {background-position: right bottom;} //alinha à direita e abaixo. Aceita também valores e porcentagens.

#### **4.10 ESTRUTURA DO LAYOUT (WIREFRAME):**

```
<div id="...">
```

```
<header>...</header>
```

```
<nav>...</nav>
```

```
<main>...</main> //recomendável que se tenha um único main na página
```

```
<footer>...</footer>
```

```
</div>
```

#### **Outras TAG semânticas:**

<section>: definir uma seção (um bloco, uma área no site)

<article>: conteúdo principal

<aside>: à parte do conteúdo principal (área de propaganda, por exemplo)

#### **TAGS não semânticas:**

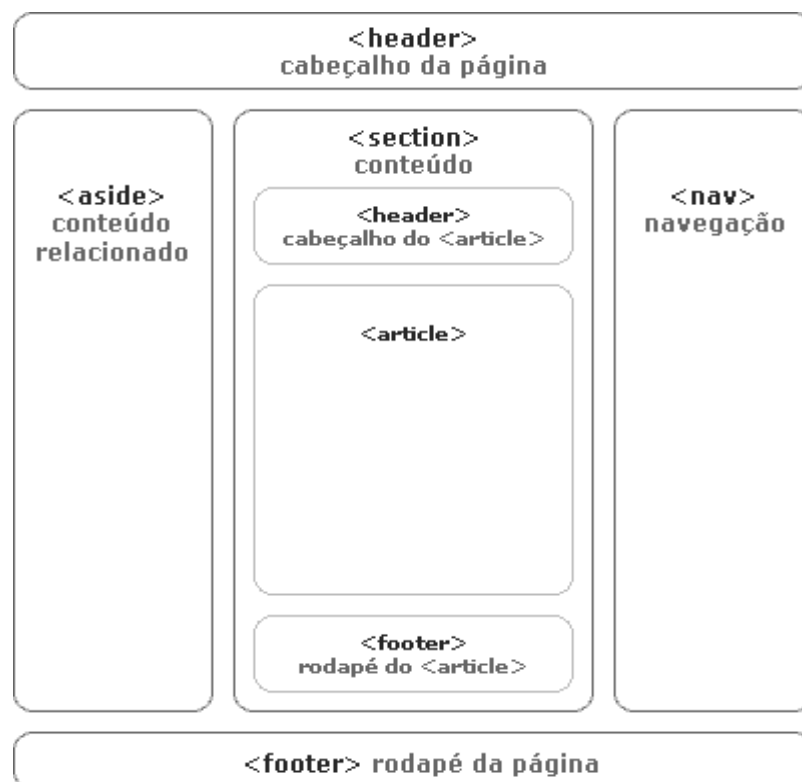
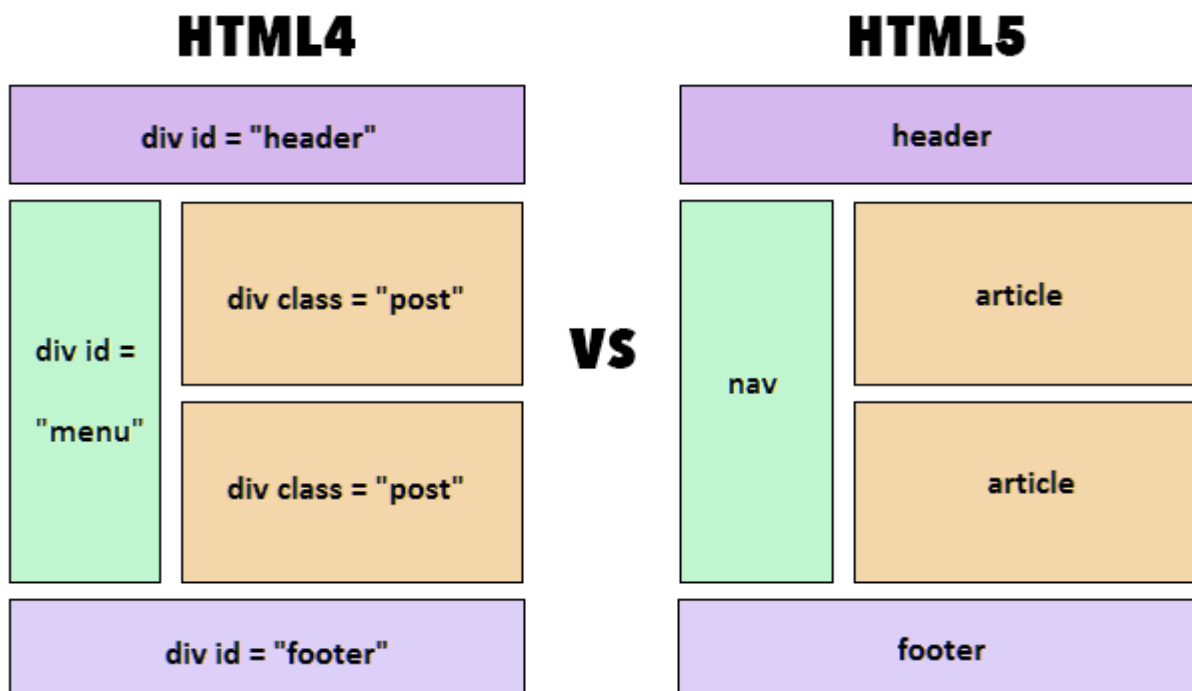
```
<div>
```

<spam>: geralmente utilizada dentro de um texto

<adress>...</adress>: Informações de contato com o autor do artigo (se no escopo de um <article>) ou da página.



Exemplos de layouts:



#### 4.11 PROPRIEDADE FLOAT E CLEAR: AULA 33

Flutuação de elementos, saindo do fluxo regular do documento.

Tomando dois <DIV> como exemplos, sendo um “pai” (container) e um “filho”: quando se aplica o FLOAT no filho, ele sai do fluxo do documento, flutuando dentro do container, e os elementos seguintes são reajustadas dentro do container.

No HTML:

```
<div id="container">
  <div id="div1">div1</div>
  <div id="div2">div2</div>
  <div id="div3">div3</div>
</div>
```

Exemplo no CSS:

```
#div1 {
  Float: left;
}
```

Pode receber os valores right, left ou none (default).

O FLOAT gera efeitos colaterais em outros elementos próximos. No entanto, aplicar a propriedade CLEAR no elemento afetado faz com que ele limpe para si os efeitos colaterais causados pelo FLOAT de outro elemento.

Assim como o FLOAT, o CLEAR também pode receber os valores:

none, left, right ou both.

```
.div2 { float: left; }
.div3 { clear:left; }
```

A CLEAR deve ser usada com o mesmo valor do FLOAT, exceto se utilizar **both**, que limpará qualquer que tenha sido o FLOAT utilizado.

**Dica:** se utilizar o mesmo valor de FLOAT para todas as <DIV>, elas tendem a se encaixar horizontalmente, flutuando fora do container. Porém, se a janela for minimizada e diminuída, os <DIV> que não cabem são jogados para baixo.

**Dica importante:** quando houver uma <DIV> container e elementos internos a ela, cuja altura extrapola a altura do container, e passa a flutuar por cima, uma solução é criar um objeto dinamicamente (via :after) e limpar a flutuação com o clear.

```
.clearfix::after {
  Content: ' ';
  Display: block;
  Clear: both;
}
```

Poderia, como “alternativa menos elegante”, utilizar **overflow: auto**; na <DIV> container.

#### **4.12 PROPRIEDADE DISPLAY:**

Elemento com comportamento em nível de bloco (block): cada elemento gera uma quebra de linha. Assim, os elementos são agrupados na vertical, um embaixo do outro. Pode-se aplicar as propriedades do Box Model. Exemplos: <header>, <footer>, <ul>, <ol>, <div>, <p>.

Elemento em nível de linha (inline): os elementos são agrupados horizontalmente, lado a lado. Exemplos: <a>, <b>, <center>, <img>, <span>.

Inline: atribuindo o valor inline na propriedade display faz com que o elemento se comporte como objeto de linha.

Block: atribuindo o valor block na propriedade display atribui comportamento de bloco para elemento em nível de linha (adicionar uma quebra de linha). Exemplo: `b { display: block; }`  
Se aplicar uma propriedade de Model Box (como width, height, padding e margin) em elemento de linha, ou não funcionará, ou funcionará de forma incoerente.

Inline-block: Aplicar propriedades do Model Box a elementos de linha. O elemento se comporta como se fosse de linha (sem adicionar quebra de linha), mas aceitará as propriedades de um objeto de bloco.

None: Retira do HTML o elemento (exclui da página).

#### **4.13 PROPRIEDADE WHITE-SPACE:**

Quando imagens grandes são alinhadas horizontalmente pelo **display: inline-block**; mas não cabem na mesma linha, as linhas são quebradas e as imagens passam a ser alinhadas verticalmente. A propriedade **white-space: nowrap**; força que fiquem obrigatoriamente na mesma linha. Como consequência, geram barra de rolagem na horizontal.

#### 4.14 DICAS PARA CONSTRUIR UM LAYOUT DE EXEMPLO

Etapas:

Estilo CSS salvo em arquivo à parte, e vinculado ao HTML pela TAG `<link rel="stylesheet" href="...">`.

1ºPasso: desenhar o rascunho do layout e identificar cada uma de suas partes;

2ºPasso: Escrever o HTML “cru”;

3ºPasso: incluir os CSS em arquivo apartado e fazer vinculação com o HTML.

Modelo base:



DICA: Fazer menu horizontal (AULA 31).

Faz uma lista `<ul><li>` com links.

Converte `<li>` para inline (deixa os links na horizontal).

Converte `<a>` para *block*. Adiciona borda, *padding* e *background-color*.

Coloca toda a formatação que desejar.

Para dar o efeito hover, utiliza-se a sub-classe. Exemplo:

```
.hero nav a:hover{
  background-color: rgba(0,0,0,0.3);
}
```

**Dica:** Para controlar a posição vertical de um menu contido em uma <nav> que está dentro de um <header>, pode-se colocar uma *border* transparente no <header> e controlar a margin-top da <nav>.

- Construindo os cards laterais (com 3 cards):

1. No HTML, criar uma seção (<section>), contendo 3 <div>. Dentro de cada <div>, inserir uma imagem e um parágrafo. Na <section> definia uma classe (.content-section), assim como em cada <div>, sendo estas todas pertencentes à mesma classe (.card).

2. No CCS, implemente a .content-section da forma que preferir. Na .card, defina largura de 32%, float:left e box-sizing como border-box. Em vez do FLOAT, poderia se utilizar o display inline-block, porém ele gera espaços entre os cards, que seria indesejável no ajuste dos layouts. No entanto, ao usar o FLOAT, causa efeito colateral na altura da <section>, desajustando o layout. Para correção, adiciona-se overflow:auto; na .content-section (que a seção mãe contendo todos os cards, e que é prejudicado pelo FLOAT dos seus filhos).

3. Defina as imagens filhas de cada <div> (.card img{...}) como largura de 100%, para que o tamanho da imagem se ajuste ao tamanho da <div>, que havia sido definida anteriormente como 32% de largura.

4. Para alinhar horizontalmente, defina duas classes para as duas primeiras <div> e inclua margem direita de 2%.

No HTML:

```
<section class="content-section">
    <div class="card card-1">
        
        <p>Seletores</p>
    </div>
    <div class="card card-2">
        
        <p>Posicionamentos</p>
    </div>
    <div class="card">
        
        <p>Fontes e Ícones</p>
    </div>
</section>
```

No CSS:

```
.content-section{
    background-color: white;
    margin-top: 10px;
    padding: 10px;
    overflow: auto;
}
```

```
.card {
    width: 32%;
    float: left;
    box-sizing: border-box;
}
```

```
.card-1, .card-2{
    margin-right: 2%;
}
```

```
.card img{
    width: 100%;
    display: block;
    /*Importante para evitar um espaçamento entre a
    imagem e a descrição que virá abaixo. Todo objeto
    INLINE gera espaçamento.*/
}
```

Para formatar as legendas:

```
.card p{
  background-color: rgba(0,0,0,0.7);
  margin: 0px;
  padding: 12px;
  color: white;
  text-transform: uppercase;
  letter-spacing: 2px;
  font-size: 10px;
}
```

#### 4.15 PROPRIEDADE POSITION:

- **Static** (default): ordem no HTML é a ordem na tela.

As posições relative, absolute e fixed requerem algumas propriedades complementares top, bottom, left, right. Elementos que possuem uma dessas três propriedades são chamados de **elementos posicionados**.

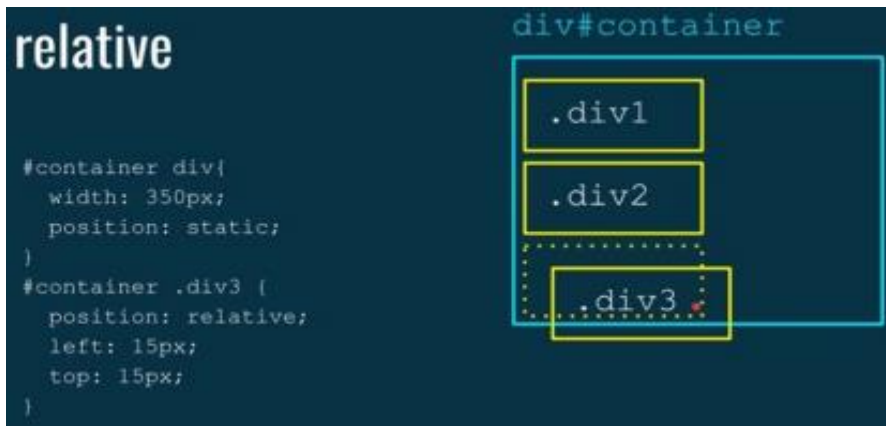
- **Relative**: Altera a posição do elemento em qualquer direção, em relação a ele mesmo, em um plano XY (horizontal-vertical). Exemplo:

```
.div3 {
  Position:relative;
  Left: 15px;
  Top: 15px;
}
```

Caminha a <div> 15px para a direita (positivo no eixo X) e para baixo em 15px (positivo no eixo Y).

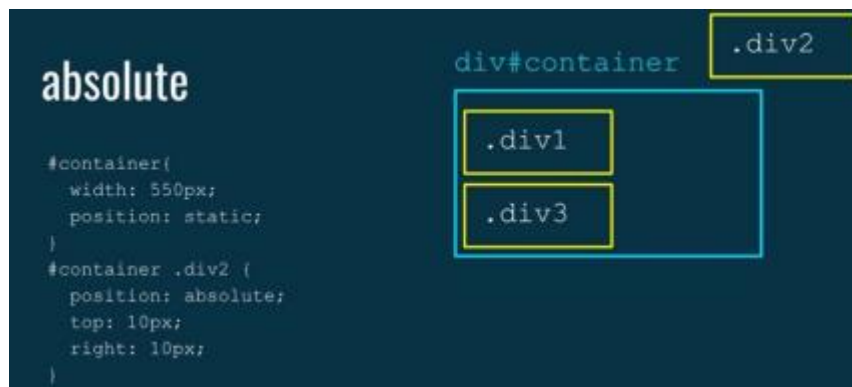
#### **Atenção:**

Embora o elemento se desloque visualmente, a posição inicial é preservada em relação ao content e em relação aos demais <div> irmãos.

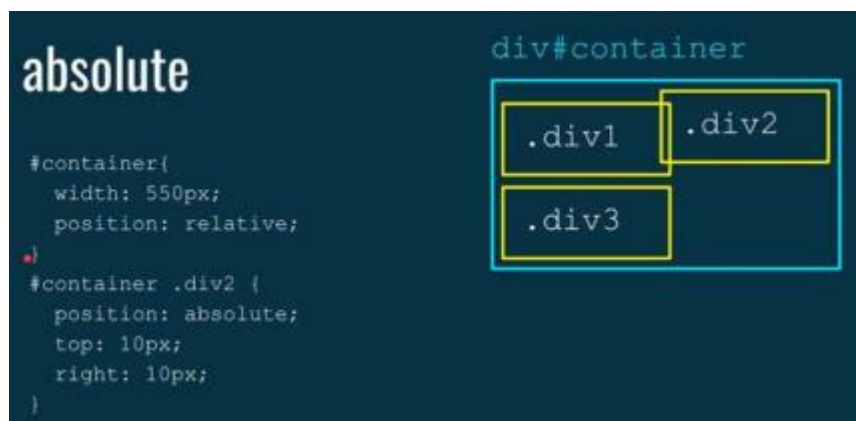


- **Absolute**: Altera a posição do elemento, efetivamente, e define nova posição no documento (<body>), em relação ao primeiro elemento **pai posicionado, ainda que não seja o pai imediato.**

Exemplo abaixo: a “.div2” não possui elemento pai posicionado, assume posição em relação ao <body>. Se a id #container fosse posicionada (tivesse position como relative, absolute ou fixed), a nova posição de <div2> seria em relação ao container, e se manteria dentro dele.



Exemplo quando <div2> possui elemento pai posicionado:





**- Fixed:** Assume uma posição fixa, em relação à janela do browser (não em relação ao corpo do documento). No caso do <header> lateral com o <footer> sobreposto, precisa colocar ambos como fixed. O <footer> sendo escrito posteriormente ao <header> terá posição mais avançada. Absolut e fixed alteram o fluxo dos elementos na página.

#### 4.16 PROPRIEDADE Z-INDEX:

Ordenar os elementos, dentro de um fluxo ou sobreposições. Z-INDEX maior fica por cima do ZINDEX menor.

#### 4.17 LISTAS:

- Linhas aninhadas: uma lista dentro da outra.

```
<ul>
  <li>item 1</li>
  <li>item 2
    <ol>
      <li>item 2.1</li>
      <li>item 2.2</li>
    </ol>
  </li>
  <li>item 3</li>
</ul>
```

- Listas de definição: Dicionário.

```
<dl>
```

```
<dt></dt>          //termo a ser descrito (definition term)
```

```
<dd></dd>          //significado do termo (definition description)
```

```
<dt></dt>          //termo a ser descrito (definition term)
```

```
<dd></dd>          //significado do termo (definition description)
```

```
...
```

```
</dl>
```

Tanto é possível existir um termo com mais de uma definição,  
ou mais de um termo com uma única definição.

# Listas de definição

```
<dl>
  <dt></dt>
  <dd></dd>
  <dt></dt>
  <dd></dd>
</dl>
```

```
<dl>
  <dt></dt>
  <dd></dd>
  <dd></dd>
  <dd></dd>
</dl>
```

```
<dl>
  <dt></dt>
  <dt></dt>
  <dt></dt>
  <dd></dd>
</dl>
```

- Fazendo aparecer uma sub-lista, quando passar o mouse sobre um item da lista:

```
<style>

  li ul{
    display: none;
  }

  li:hover{
    background: #ccc;
  }
  li:hover ul{
    display: block;
  }

</style>
```

1. Oculta da página toda <ul> que estiver dentro de uma <li> (oculta a lista aninhada);
2. Ao passar o mouse sobre um item da lista, o background fica cinza;
3. Reexibe a <ul> escondida, quando passar o mouse sobre uma <li>.

Interessante: o evento *hover* pode ser utilizado em qualquer elemento, e não apenas no *a*.

- Fazendo uma animação de lista, como se fosse um menu com sub-menu:

```
li ul{
  height: 0;
  overflow: hidden;
  transition height .5s;
}

li:hover{
  background: #ccc;
}
li:hover ul{
  height: 50px;
}
```

1. Em qualquer <ul> que estiver dentro de uma <li> (lista aninhada), define a altura como 0, deixa escondida (overflow: hidden) e inclui um efeito de transição com duração de 0,5s.
2. Quando passar o mouse sobre qualquer <li>, exibe um fundo cinza.
3. Quando passar o mouse sobre uma <li>, qualquer <ul> filha dela assume altura 50px, anulando o overflow e fazendo com que a <ul> volte a ser exibida.

#### **4.18 PROPRIEDADE OVERFLOW:**

Serve para dar tratamento a um conteúdo que excede um determinado elemento (parte de um texto que ultrapassar uma <div>, por exemplo). Utilizado no container.

*Valores:*

- Visible (default): Exibe a parte excedente.
- hidden: Esconde a parte excedente.
- scroll: gera scrolls horizontais e verticais no elemento, para exibir o excedente.
- auto: gera scroll apenas e for necessário.

#### **4.19 TABELAS: Aula 45**

Tabela com uma linha e duas colunas:

```
<table>
```

```
    <tr>
```

```
        <td></td>
```

```
        <td></td>
```

```
    </tr>
```

```
</table>
```

- Atributos de mesclagem: Colspan e Rowspan:
- Colspan: expande uma coluna.

## Principais tags

```
<table>
  <tr>
    <td colspan="2">1A</td>
  </tr>
  <tr>
    <td>2A</td> <td>2B</td>
  </tr>
  <tr>
    <td>3A</td> <td>3B</td>
  </tr>
</table>
```

table

1A	
2A	2B
3A	3B

- Rowspan: expande uma linha.

## Principais tags

```
<table>
  <tr>
    <td rowspan="2">1A</td>
    <td>1B</td>
  </tr>
  <tr>
    <td>2B</td>
  </tr>
  <tr>
    <td>3A</td> <td>3B</td>
  </tr>
</table>
```

table

1A	1B
	2B
3A	3B

- Cabeçalhos de tabelas: <thead> e <tbody>

## Table header e Table body

```
<table>
  <thead>
    <tr>
      <th>1A</th> <th>1B</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>2A</td> <td>2B</td>
    </tr>
  </tbody>
```

table

1A	1B
2A	2B
3A	3B

Dentro de <thead>...</thead> não existe <td></td>, que são substituídas por <th></th>.

- Rodapé em tabelas: <tfoot>. Atenção: deve vir logo após de <thead> e antes de <tbody>.

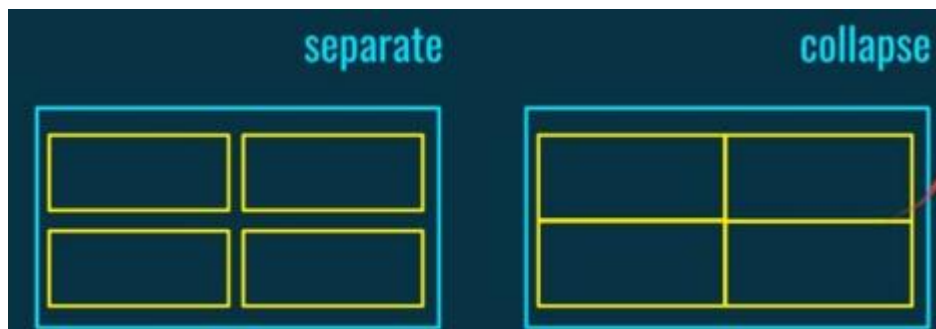


- Propriedade border-collapse: retira o padding das bordas das tabelas.

Valores:

- separate (default): mantém os espaços entre as colunas e linhas.

- collapse: retira os espaços.



- <caption>...</caption>: título da tabela. Também importante para acessibilidade.

#### 4.20 FORMULÁRIOS:

<form>...</form>: TAGs que especificam o início e o fim do formulário.

Atributos:

action="pagina.php": Arquivo no servidor que irá processar o formulário.

method="GET ou POTS": método de envio dos dados para processamento (GET passa os valores através da URL).

<input>: Entrada de dados.

Atributos:

- type="text" name="nome\_do\_campo": entrada de texto normal.

- type="radio": opções de única seleção. Todas as opções precisam ter o mesmo nome, diferenciados apenas pelo atributo value, para identificar qual opção foi selecionada. Para que a seleção ocorra não apenas quando se clica na caixa circular do radio, mas também na palavra que se refere, deve-se utilizar o <label>.

Exemplo:

```
<p>Sexo:</p>
<label>
  <input type="radio" name="sexo"
    value="M">
  Masculino
</label>
<label>
  <input type="radio" name="sexo"
    value="F">
  Feminino
</label>
```

- type="checkbox": Funcionamento semelhante ao radio, mas não necessita a condição de possuírem o mesmo name. A diferença é que o usuário poderá selecionar mais de uma opção.
- type="range": uma barra de seleção. Precisa ter os atributos min e max, e também step.
- type="submit" value="texto do botão": botão de enviar.

Exemplo: Busca no Google.

```
<form action="http://www.google.com.br/search" method="GET" target="_blank">
  <input type="text" name="q">
  <input type="submit" value="Pesquisar">
</form>
```

Outros tipos de entrada de dados (valores para o atributo type):

Tipos de Entrada de dados	
input	input (HTML5)
• text	• email
• password	• tel
• checkbox	• url
• radio	• search
• file	• number
• hidden	• range
• submit	• date
• button	• time
• reset	• color

**<textarea>texto a ser exibido por default</textarea>**: Cria uma caixa de digitação de texto. Os atributos `cols="..."` e `rows="..."` definem o tamanho da área, em linhas e colunas. Através de CSS, pela propriedade `resize:none`; pode-se desabilitar a opção de redimensionamento pelo usuário. Via CSS, também se pode alterar altura e largura, com *width* e *height*.

O atributo **readonly** inibe a edição do texto. O atributo **disabled** deixa desabilitado, e não apenas inibe a edição.

**<select><option>...</option></select>**: Cria uma caixa de seleção.

**<button>rótulo do botão</button>**:

Outros atributos:



Autofocus: faz com que o foco já se inicie no objeto do formulário.

Exemplo: `<input type="text" name="login" autofocus>`.

Min e max: define valores mínimos e máximos aceitos pelo campo, quando for do tipo número ou data (a atribuição nos atributos min e max para datas segue o padrão YYYY-MM-DD).

Multiple: a ser definido no `<select>`, permite a seleção de mais uma `<option>`, quando selecionado em conjunto com CTRL.

Placeholder: hint, dica que começa preenchendo um campo.

step: pode ser utilizando em tipo range ou number, para especificar um intervalo entre as opções.

Dica ao utilizar evento :focus em campos de formulário, para realçar a borda: definir os campos com borda transparente, e incluir a cor na definição do CSS. Se aparecer conjuntamente uma borda azul padrão do browser, incluir a propriedade `outline: none`;

Implementando o CSS por atributo: `seletor[atributo] {...}` . Exemplo:

```
Input[type="text"] {
```

```
Background-color: blue; //atribui background apenas para <input type="text">
```

```
}
```

- TAGS com atribuição de semântica:

**<label>...</label>**: Vincula um texto a um campo do formulário, atribuindo semântica.

```
<label for="vínculo_label_id">Texto a ser exibido</label>
```

```
<input type="text" name="q" id="vínculo_label_id">
```

**<fieldset>...</fieldset>**: Vincula um grupo de objetos relacionados entre si a alguma finalidade (cria um grupo lógico). Na tela, os campos do formulário são enquadrados por um frame.

Exemplo:

O diagrama mostra um exemplo de formulário de login. À esquerda, há um bloco de código HTML com as seguintes tags destacadas por retângulos vermelhos: `<fieldset>`, `<legend>Login</legend>`, `<label for="log">User:</label>`, `<input type="text" name="text" id="log">`, `<label for="pass">Password:</label>`, `<input type="password" name="pass" id="pass">`, `</fieldset>` e `<input type="submit" value="Login">`. À direita, há uma representação visual do formulário. O título "login" está em um retângulo vermelho. O formulário é contido em um retângulo azul claro com uma borda amarela. Dentro dele, há o rótulo "Usuario" seguido de um campo de texto amarelo, o rótulo "Senha" seguido de um campo de senha amarelo, e um botão "Login" amarelo.

- CSS EM FORMULÁRIOS:

Sub-classe para evento:

```
.classe ou #id: focus {
```

```
    Formatação quando o item do formulário recebe o foco.
```

```
}
```

#### 4.21 EFEITOS INTERESSANTES COM CSS:

- Border-radius: formatar campos arredondados para uma <DIV>. Para se obter uma <div> com formato de esfera perfeita, o border-radius deve ser metade da largura e altura de um quadrado.

```
Div {
```

```
    Border-radius: borda_sup_esquerda(px), sup_direita, inf_direita, inf_esquerda [/dist.na vert. ;
```

```
}
```



- Box-shadow: Inclui sombra na <DIV>. Aceita valores positivos (sombra à direita ou abaixo) ou negativos (sombra à esquerda ou acima). Aceita mais de uma sombra, cujos valores são separados por vírgulas. O parâmetro **inset** define que o sombreado será colocado na parte interna da borda.

Box-shadow: [inset] desloc\_eixoX(px) desloc\_eixoY(px) esmaçamento(px) [tamanho(px)] cor, [valores da sombra 2], [valores da sombra3];

Exemplo:

```
Div {
    Box-shadow 10px 10px 0 black, 10px 10px 10 red;
}
```

- Text-shadow: Parecido com box-shadow, porém aplicada à textos. Porém, não existe a possibilidade de aumentar ou reduzir o tamanho da sombra. Também aceita múltiplas sombras.

- Gradientes lineares: É aplicado sobre um background-image.

Background-image: linear-gradient([sentido1], [sentido2]..., cor1 [ponto onde começa em px ou %], cor2 [ponto onde começa], cor3 [ponto onde começa] ...);

Sentido pode assumir: to top, to right, to bottom (default), to left, combinações para diagonal (to bottom left, por exemplo) ou em graus (exemplo: 70deg). Aceita mais de um gradiente, assim como na shadow, separados por vírgula. Pode-se também combinar com imagem, através do background-image: url(<arquivo>);

Exemplo:

```
body, html{
    height: 100%;
}
body{
    background-image: linear-gradient(to bottom, red, blue);
}
```

- Gradientes radial: Também é aplicado sobre um background-image.

Background-image: radial-gradient([posição], cor1 ponto onde começa em px ou %], cor2 ponto onde começa em px ou %], cor3 ponto onde começa em px ou %], ...);

As cores podem ser definidas pelo nome ou pela função rgb/rgba.

A posição pode ser:

- *circle*
- *ellipse*
- *Closest-side at X% Y%*
- *Farthest-side at X% Y%*
- *Closest-corner at X% Y%*
- *Farthest-corner at X% Y%*
- Gradientes repetitivos: Também é aplicado sobre um background-image, e tanto linear quanto radial.

Background-image: repeating-linear-gradient([sentido1], [sentido2]..., cor1 [ponto onde começa em px ou %], cor2 [ponto onde começa], cor3 [ponto onde começa] ...);

- **Efeito interessante**: Fazer com que o gradiente mude com o rolar da tela.

```
#grupo-cores div:nth-child(16){
  background-image: linear-gradient(red, blue);
  background-attachment: fixed;
}
```

**Fontes customizadas**: Utilizada para disponibilizar uma fonte ao cliente. Lembrar de subir a fonte para o servidor.

```
@font-face {
  font-family: 'myfont';
  src: url(fonts/KaushanScript-Regular.otf);
}
#efeitos-css-texto{
  font-family: myfont, Arial, Helvetica, sans-serif;
```

Endereços para baixar fontes free: <https://www.fontsquirrel.com/> ou

<https://fonts.google.com/>. O Google Fonts pode ser utilizado como vinculação externa (copiar o link na página do Google e incluir no HTML <head>).

Endereços para baixar fontes-ícone ou imagens vetoriais: <https://fontawesome.com/>

Endereço de vinculação para uso dos ícones (inserir entre <head></head>):

```
<script src="https://kit.fontawesome.com/82384ba7af.js" crossorigin="anonymous"></script>
```

Para incluir um ícone na página, selecione o ícone desejado na página fontawesome.com e inclua no arquivo HTML.

Atenção para os tipos de fontes, que podem ser suportados de forma diferente para cada browser. As mais comuns são: .eot, .svg, .ttf, .woff.

Implementação para maiores suportes, evitando que o browser faça download de fontes não suportadas (verificação prévia).

```
@font-face {
  font-family: 'MyWebFont';
  src: url('webfont.eot'); /* IE9 Compat Modes */
  src: url('webfont.eot?#iefix') format('embedded-opentype'), /* IE6-IE8 */
  url('webfont.woff2') format('woff2'), /* Super Modern Browsers */
  url('webfont.woff') format('woff'), /* Pretty Modern Browsers */
  url('webfont.ttf') format('truetype'), /* Safari, Android, iOS */
  url('webfont.svg#svgFontName') format('svg'); /* Legacy iOS */
}
```

#### 4.22 TRANSIÇÕES SUAVES:

- Transition-property: propriedade que será alterada por algum evento (como hover, por exemplo). Se definir o valor **all**, irá animar todas as propriedades que tiverem alteração no evento.

Atenção: A transição do sublinhado do link não é reconhecida pelo text-decoration. Deve-se utilizar a propriedade border-bottom e border-bottom-color: transparent;

```
.main-header-title a{
  text-decoration: none;

  border-bottom: dotted 1px rgba(255, 255, 255, 0.6);
  color: inherit;
  transition-property: color, border;
  transition-duration: 0.5s, 0.5s;
}

.main-header-title a:hover{
  border-bottom-color: transparent;
  color: rgba(67, 162, 134, 0.6);
}
```

- Transition-duration: duração da transição (ex.: 1s).

- Transition-timing-function: aceleração da animação.

Linear: velocidade constante.

Ease: desacelera no final.

Ease-in: acelera no final.

Ease-out: desacelera.

Ease-in-out: acelera e desacelera.

Cubic-bezier: Coordenadas X1, Y1 (ponto inicial) e X2, Y2 (ponto final). O movimento pode ser visualizado pelo site <https://cubic-bezier.com>.

- Transition-delay: retardo para início da animação (ex.: 0.5s).

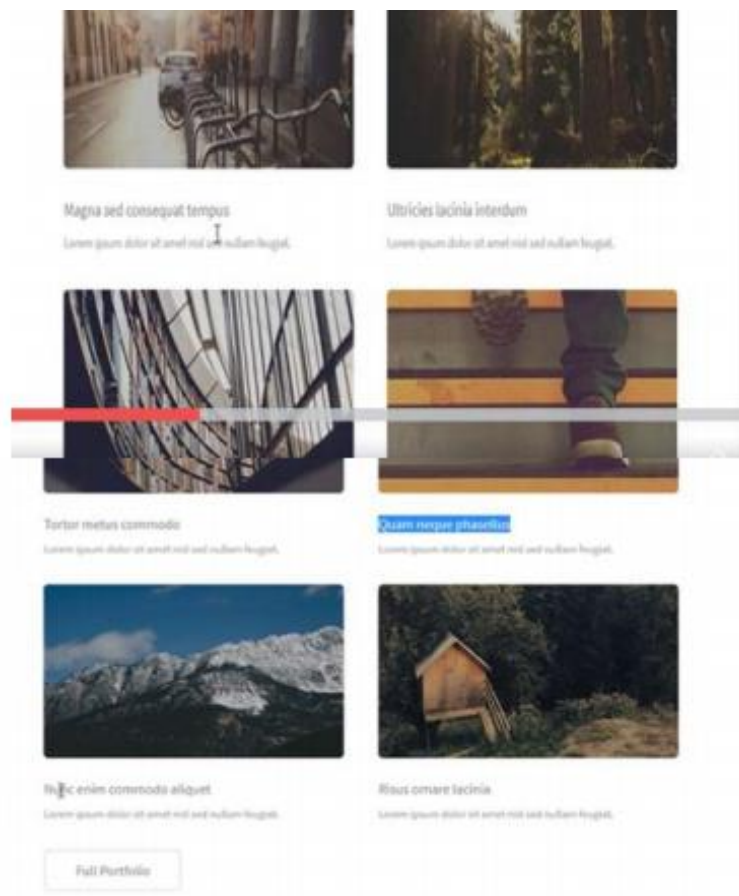
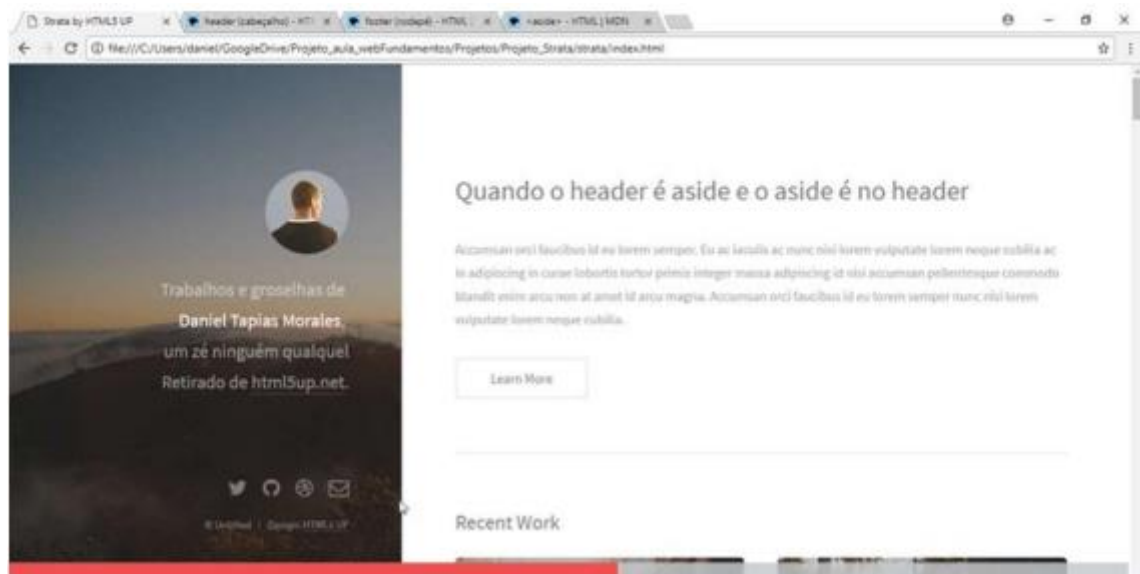
Pode-se acumular transições para propriedades distintas. Nesse caso, separa-se por vírgulas.

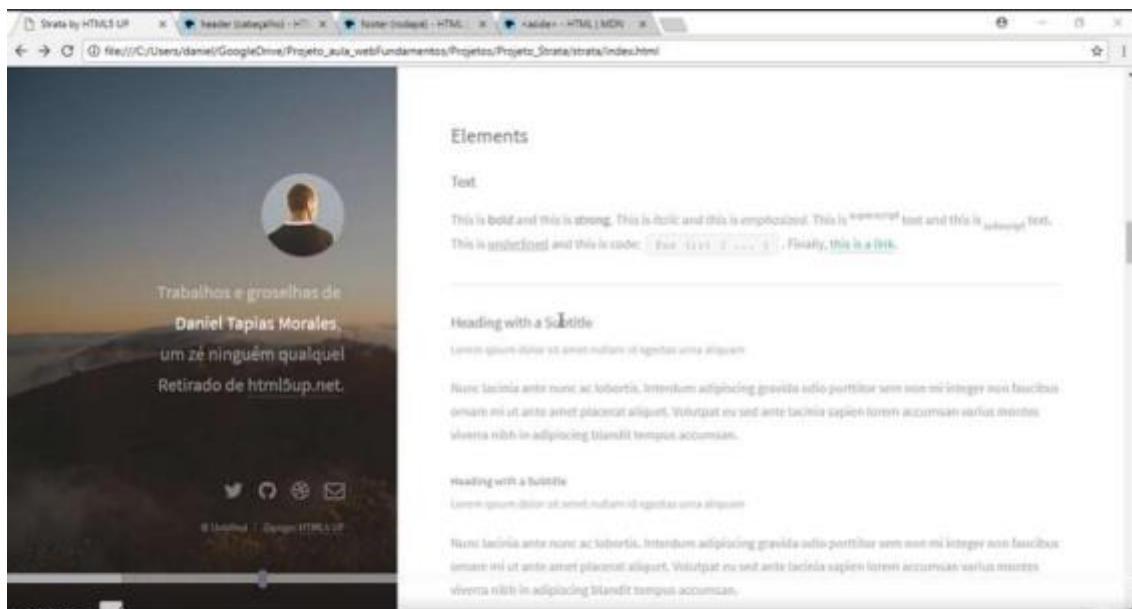
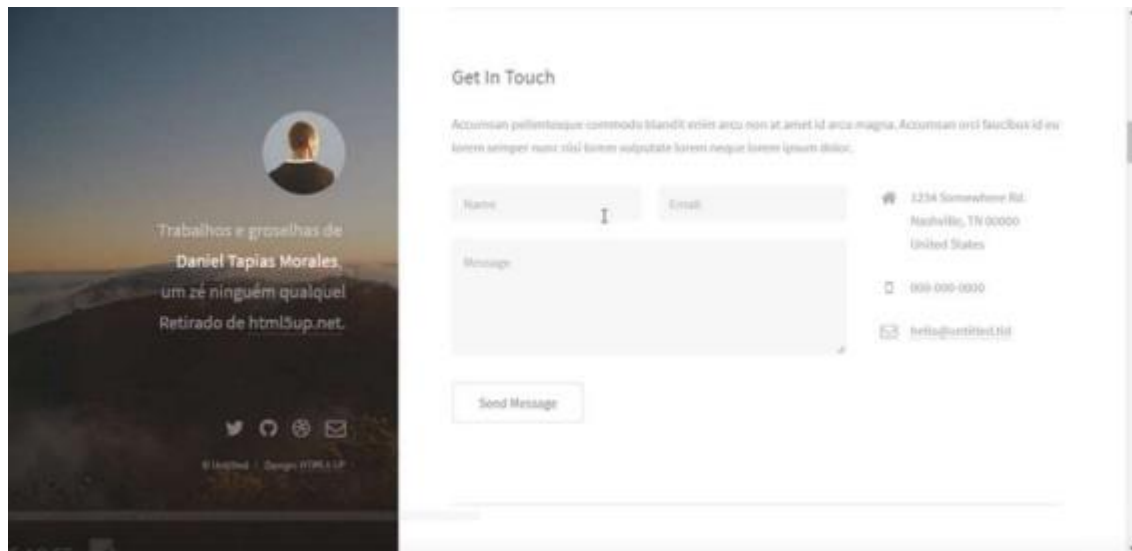
```
div{
  width: 100px;
  height: 100px;
  background-color: orange;

  transition-property: width, height;
  transition-duration: 1s, 3s;
  transition-timing-function: cubic-bezier(.67,-0.38,.5,1.5);
  transition-delay: 0s, .8s;
}

div:hover{
  width: 400px;
  height: 400px;}
```

## 4.23 DICAS NO PROJETO STRATA:





## ➤ CONSTRUINDO O MENU LATERAL FIXO, UTILIZADA COMO <HEADER>

No HTML:

```
<header class="main-header">
  <a href="http://html5up.net" class="main-header-link">
    
  </a>
  <h1>Um título qualquer</h1>
</header>
```

No CSS: Para fazer o alinhamento é melhor trabalhar com posição e largura dos objetos que compõem o layout, em vez de propriedades display ou float.

```
body{
  line-height: 1.7;
```

```

    font-
family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sa
ns Unicode', Geneva, Verdana, sans-serif;
    color: #666666;
}
.main-header{
    position: fixed;    /*Para que o <header> lateral se mantenha fixo co
m scroll da página*/
    top: 0px;
    left: 0px;
    height: 100%;
    width: 35%;
    padding: 164px 54px 72px 54px;
    box-sizing: border-box;
    background-image: url(../images/overlay.png), url(../images/bg.jpg);
    background-position: left top;
    background-size: auto, cover;
}

```

## ➤ COLOCAR ELEMENTOS EM DUAS COLUNAS

1. Cria uma <DIV> como linha (<DIV class="row">)
2. Cria duas <DIV> internas, como duas colunas.

No HTML:

```

<div class="container">
    <div class="li-col1">
        <ul>
            <li>Lista 1</li>
            <li>Lista 2</li>
            <li>Lista 3</li>
        </ul>
    </div>
    <div class="li-col2">
        <ol>
            <li>Lista 1</li>
            <li>Lista 2</li>
            <li>Lista 3</li>
        </ol>
    </div>
</div>

```

No CCS:

```

.li-col1{
    width: 50%;
    float: left;
}

```

```

    margin: 0px;
    padding: 0px;
}

.li-col2{
    width: 50%;
    float: left;
    margin: 0px;
    padding: 0px
}

```

➤ Alinhamento do formulário Get in Touch:

**Get In Touch**

Accumsan pellentesque commodo blandit enim arcu non at amet id arcu magna. Accumsan orci faucibus id eu  
 lorem semper nunc nisi lorem vulputate lorem neque lorem ipsum dolor.

Name	Email	1234 Somewhere Rd. Nashville, TN 00000 United States 000-000-0000 hello@untitled.tld
Message		

Send Message

➤ Colocar um elemento invisível:

Definir uma classe, com as seguintes propriedades:

```

.classe {
    Width: 0;
    Height: 0;
    Overflow: hidden;
    Visibility: hidden;
    Display: block;
}

```



#### 4.24 SELETORES COMPOSTOS:

DOM: Estruturas de parentesco entre os elementos (“árvore genealógica”), representando os elementos do documento.

Descendência: Vínculos entre os elementos. Exemplo:

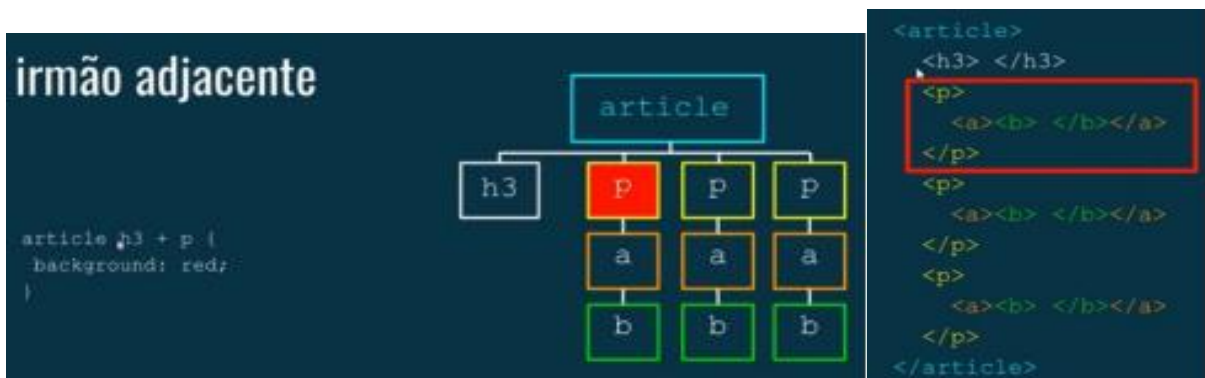
`article a { ... }` //Afeta todas as TAGs `<a>` descendentes de `<article>`.

Filho direito: Selecionar apenas os elementos diretamente filhos.

`article p > b {...}` //afeta apenas o `<b>` diretamente filho de `<p>`, que por sua vez seja filho de `<article>`

Irmão adjacente: Afeta **apenas** o próximo irmão imediato.

`article h3 + p {...}` //afeta o primeiro `<p>` após `<h3>`, quando ambos são filhos de `<article>`



Próximos irmãos: Afeta **todos** os irmãos imediatos.

`article h3 ~ p {...}` //afeta todos os `<p>` após `<h3>`, quando ambos são filhos de `<article>`



## 4.25 SELETORES DE ATRIBUTOS:

Utiliza-se os colchetes.

### Atributo qualquer:

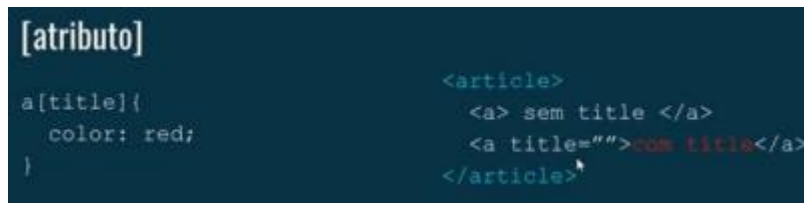
Seletor[atributo] {...} //seleciona todo elemento que tiver o atributo especificado.

Exemplo:

a[title] { //todas as tags <a> que tiver o atributo title, independentemente do valor, terão cor vermelha. As outras, não.

color: red;

}



[atributo]

```
a[title]{
  color: red;
}
```

```
<article>
  <a> sem title </a>
  <a title="">com title</a>
</article>
```

### Atributo Específico Exato:

Seletor[atributo="valor"] {...} //seleciona todo elemento que tiver o atributo = "valor".

Exemplo:

a[alt="teste"] { //apenas as tags <a> que tiver o alt="teste" terão cor vermelha.

color: red;

}



[atributo=""]

```
img[alt=""]{
  border-color: red;
}
```

```
<article>
  <img alt="texto">
  <img alt="">
</article>
```

### Atributo Específico com valor inicial:

Seletor[atributo^="valor"] {...} //seleciona todo elemento cujo atributo iniciar com "valor".

Exemplo:

a[alt^="teste"] { //apenas as tags <a> que o atributo alt iniciar com "teste" terão cor vermelha.

color: red;

}

```
[atributo^="valor"]
href=
a[href^=http]{
  color: red;
}

<article>
  <a href="..."> link interno </a>
  <a href="http://">link externo</a>
</article>
```

### Atributo Específico com valor final:

Seletor[atributo\$="valor"] {...} //seleciona todo elemento cujo atributo terminar com "valor".

Exemplo:

```
a[alt^="teste"] { //apenas as tags <a> que o atributo alt terminar com "teste" terão cor
                  vermelha.
  color: red;
}
```

```
[atributo$=""]
img[src$=".png"]{
  border-color: red;
}

<article>
  
  
</article>
```

### Seletor de negação:

:not(a) {...} //Seleciona tudo que **não** for <a>

```
img:not([alt]){
  display: none;
}
```

Seleciona todas as imagens que não contenham o atributo alt.

```
p:not(.destaque){
  ...;
}
```

Seleciona todos os <p> que não possuem a classe .destaque.

## 4.26 INCLUIR CONTEÚDO DINAMICAMENTE COM CSS: CONTENT, AFTER E BEFORE

Div:before {

Content: "conteúdo"; //adiciona o texto "conteúdo" antes do conteúdo atual da  
<div>

}

Exemplo:

```
<div class="add-dyn">
  <p>Conteúdo original.</p>
</div>
```

Incluindo um pequeno quadrado vermelho dinamicamente:

```
.add-dyn{
  border: solid 1px green;
  position: relative; /*Importante para funcionar o
                     posicionamento no :before*/
}

.add-dyn:before{
  display: block;
  content: "";
  width: 20px;
  height: 20px;
  background-color: red;
  position: absolute;
  right: 50%;
  bottom: 0px;
}
```

O content: pode receber a função **url()** para incluir uma imagem.

A função **attr(href)** recupera o endereço de um link. Poderia ser qualquer atributo, e não apenas o href.

Exemplo:

```
a:after {
  content: "(" attr(href) ";
display: inline-block;
}
```

Vai exibir o endereço **após** a criação de um link.

#### 4.27 PSEUDO-CLASSES E PSEUDO-ELEMENTOS:



<code>:checked</code>	<code>:last-child</code>	<code>:only-of-type</code>
<code>:disabled</code>	<code>:last-of-type</code>	<code>:out-of-range</code>
<code>:empty</code>	<code>:link</code>	<code>:read-only</code>
<code>:enabled</code>	<code>:not()</code>	<code>:read-write</code>
<code>:first-child</code>	<code>:nth-child()</code>	<code>:required</code>
<code>:first-of-type</code>	<code>:nth-last-child()</code>	<code>:target</code>
<code>:focus</code>	<code>:nth-last-of-type()</code>	<code>:valid</code>
<code>:hover</code>	<code>:nth-of-type()</code>	<code>:visited</code>
<code>:invalid</code>	<code>:only-child</code>	

<https://developer.mozilla.org/pt-BR/docs/Web/CSS/Pseudo-classes>

Pseudo-classes começam com `:` e pseudo-elementos com `::`. Pseudos elementos também podem ser escritos com `:`, que é a forma suportada por browsers mais antigos.



- `::after`
- `::before`
- `*::first-letter`
- `::first-line`
- `::selection`
- `::placeholder`

Interessante: `::selection`, que é a seleção do usuário e *placeholder* também podem ser formatados.

```
p::selection{
  background-color: red;
  color: yellow;
}
```

Pseudo-classes e pseudo-elementos podem ser combinados.

```
p:empty:before{
  content: "paragrafo vazio nao dá, ne!";
}
```

-Outras sub-classes (relacionadas a lista, linhas de tabelas ou quaisquer elementos com hereditariedade):

:focus: quando o elemento recebe o foco.

:first-child: pega a última linha de uma lista (pode ser <li> ou <tr>, por exemplo).

:last-child: pega a última linha de uma lista (pode ser <li> ou <tr>, por exemplo).

:nth-child(N): pega a linha N de uma lista (pode ser <li> ou <tr>, por exemplo). As constantes **odd** e **even** representam os números **ímpares** e **pares**, respectivamente.  $N=2n$  significa pegar os elementos múltiplos de 2. **1n**, os múltiplos de 1, **3n** os múltiplos de 3 e assim sucessivamente.

Pode-se utilizar fórmulas, do tipo **3n-1**.

Exemplos:

```
Table tbody tr:first-child td {
  Border-top: solid 1px black;
}
```

```
Table tbody tr:nth-child(odd) td {
  Background-color: red;    //inclui background nas linhas ímpares de uma tabela
}
```

:nth-of-type(N): Afeta o elemento N de determinado tipo. Exemplo:

```
Div p:nth-of-type(5) {      //atinge o 5º element <p>, que seja filho de <div>.
...
}
```

Quando não for um elemento de lista, o N é o número do elemento no <body>. Por exemplo: se tem:

```
<body>
  <a href="#">link</a>
  <p>Parágrafo</p>
  <hr>
  <article>
    <p>Parágrafo do article</p>
  </article>
</body>
```

Para pegar o <article>, a CSS deve ser `article:nth-child(5) {...}`

#### 4.28 PROPRIEDADE TRANSFORM 2D:

- translate(X,Y): mover o objeto na tela (alterar a posição). Pode-se alterar na horizontal (translateX) ou na vertical (translateY). A propriedade transform: translate() reposiciona nos dois eixos. Semelhante ao position:relative; Ambos deslocam o elemento, mas mantém reservada a posição inicial. A diferença será no uso do valor em porcentagem. Enquanto o *translate* leva em consideração sua própria proporção para o cálculo da porcentagem, o *position* leva em consideração a dimensão do elemento pai. A alteração da posição não altera a ordem do elemento no fluxo do documento.
- skew(Ndeg): Distorce o elemento, no grau informado. Transforma um quadrado em um trapézio, por exemplo. Aceita o skewX(Ndeg) e o skewY(Ndeg).
- scale(N): Altera a proporção (tamanho), em N vezes. Existe o scaleX(n) e scale(Y).
- rotate(Ndeg): Rotaciona o elemento em N graus.
- transform-origin: muda a posição do eixo de rotação da propriedade *rotate*. Pode assumir as posições top, right, bottom, left e center.

Pode-se utilizar os dois valores simultaneamente, separados por espaço.

```
.div6-interna{  
  background: yellow;  
  transform: rotate(30deg) translate(50%);  
}
```

Neste caso, a rotação é feita primeiro. Assim, o reposicionamento se dá em um eixo X já rotacionado.

## 4.29 CRIANDO UM CHECKBOX ESTILIZADO:

### - No HTML:

```
<label>
  <input type="checkbox">
  <span class="chk-spam"></span>
  Esporte
</label>

<label>
  <input type="checkbox">
  <span class="chk-spam"></span>
  Política
</label>

<label>
  <input type="checkbox">
  <span class="chk-spam"></span>
  Religião
</label>
```

### - No CCS:

```
body{
  font-family: sans-serif;
}

label{
  display: block;
}

.chk-spam{
  display: inline-block;
  border: solid 1px #bbbbbb;
  height: 18px;
  width: 50px;
  border-radius: 15px;
  background-color: #eeeeee;
}

input[type="checkbox"]{
  width: 0px;
  height: 0px;
  overflow: hidden;
  /*visibility: hidden;*/
  opacity: 0;
}

input[type="checkbox"] + .chk-spam::before{
  content: " ";
  display: block;
  width: 15px;
  height: 15px;
  border: solid 1px yellow;
  border-radius: 50%;
  background-color: red;
  transition: all 1s ease;
}

input[type="checkbox"]:checked + .chk-spam::before{
  transform: translateX(30px);
  background-color: blue;
}

input[type="checkbox"]:focus + .chk-spam{
  /*exibir borda no focus*/
  /*border: solid 1px blue; OU*/
  box-shadow: 0 0 0 2px blue;
}

</style>
```



## 5. RWD – Responsive Web Design (Seção 10 – Media Queries):

Baseado em 3 princípios:

- Grid fluída: largura em porcentagens.
  - Recursos flexíveis: conteúdos externos (vídeos de youtube, por exemplo) devem caber na tela.
  - Consultas de mídias: mídias queries.
- **Media Type:** Antigamente, dentro da TAG que incluía o .css no .html, definia-se o atributo *media*, com os valores *screen*, *print* ou *handheld*, para especificar que tipo de mídia iria receber aquele css.

```
<link href="tela.css"
rel="stylesheet" media="screen">

<link href="impressao.css"
rel="stylesheet" media="print">

<link href="mobile.css"
rel="stylesheet" media="handheld">
```

A aplicação consultava o tipo de mídia, e executava o CSS determinado.

- **Media Queries:** Em vez de consultar o tipo de mídia, consulta-se alguma característica do dispositivo, tais como a quantidade de pixels.

```
@media screen and (max-width:480px){
  body {
    background: red;
  }
}

@media screen and (min-width:481px){
  body {
    background: blue;
  }
}
```

Tipos de Media Queries:

```
min-width / max-width
min-height / max-height
min-device-width / max-device-width
min-device-height / max-device-height
orientation
aspect-ratio
device-aspect-ratio
resolution
```

- **Metatag VIEWPORT**: Equalizar a resolução ao tamanho do aparelho.

Recomendável utilizar sempre !

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

### **Passo-a-passo do site responsivo (Aula 100)**

Abordagem mobile-first.

1. Escrever o HTML (esqueleto, para formatação smartphone).
2. Incluir a normalização do CSS, para padronizar a leitura dos browsers (<https://necolas.github.io/normalize.css/>). Fazer download do CSS de normalização, e incluir no <HEAD> do HTML.
3. Criar o arquivo CSS de estilos.
4. Iniciar com `@charset "UTF-8";`
5. Criar as classes e definir no HTML.
6. Dicas para incluir ícones. Utilizar o IcoMoon APP (<https://icomoon.io/app/#/select>)
  - Acessa <https://icomoon.io/>
  - Clica no botão IconMoon APP no topo da página(espera carregar os ícones);
  - Seleciona todos os ícones que deseja utilizar, e clica em “Font” ou “Generate Font”, no canto inferior.
  - Serão demonstrados os códigos de cada símbolo, como fonte, para ser inserido na página. Clicar no botão “download” e salvar na página do projeto.
  - Incluir o arquivo CSS baixado na página HTML.
  - Procurar no arquivo demo.html baixado a classe que contém o ícone desejado, para que possa ser utilizado (ex.: `<span class="icon-alarm"></span>`).
  - Se for usar o ícone como link, utiliza-se a classe dentro da TAG `<a>`, em vez da TAG `<span>`
7. Após concluir a diagramação para smartphone, inicia-se a criação das Media Queries, para adaptação a novos modelos de tela. Exemplo: Primeira alteração quando a tela for maior que 780px:

```
@media screen and (min-width: 78.75em){ /*780px*/
}
```

## 6. Imagens Responsivas:

### Atributos:

Srcset: Informa ao browser quais as dimensões das imagens. O 400w e o 800w do exemplo informa que as imagens possuem 400 e 800 pixels de largura, respectivamente. Esta parte com **w** é chamada de “*descritor*”. O src funciona como fallback (lido caso o browser não dê suporte ao atributo srcset).

**Atributo srcset**

```

```

Sizes: Informa ao browser que a imagem será carregada a 33.33% da ViewPort (1/3 da largura da tela visível).

**Atributos srcset e sizes**

```

```

No exemplo abaixo, a imagem será carregada com 1/3 do tamanho da tela se a largura da tela for maior que 480px. Se for menor que 480px, será carregado em 100% do viewport.

**Atributos srcset e sizes**

```

```

Type:



### 6.1 TAGS FIGURE E FIGCAPTION:

```
<figure>
  
  <figcaption>Legenda da Imagem</figcaption>
</figure>
```

### 6.2 TAG <PICTURE>:

Utilizada para incluir imagens diferentes, a depender do tamanho da tela.



## 7. Incorporação de Conteúdo (Áudio, Vídeo e IFRAME):

**Audio:** `<audio src="mídia.mp3" controls></audio>`

O controle áudio é específico de cada browser, que pode não dar suporte para todos os formatos de áudio. Assim, é interessante disponibilizar o áudio em mais de um formato, da seguinte maneira:

```
<audio controls [autoplay] [loop]>
  <source src="mídia.mp3" type="audio/mp3">
  <source src="mídia.ogg" type="audio/ogg">
  <p>Seu browser não suporta este recurso</p>
</audio>
```

**Video:**

```
<video controls [autoplay] [loop] [muted]>
  <source src="mídia.mp4" type="audio/mp3">
  <source src="mídia.ogv" type="audio/ogg">
  <source src="mídia.webm" type="audio/webm">
  <p>Seu browser não suporta este recurso</p>
</video>
```

**iFrame:** Abrir um quadro para inserir conteúdo externo (de outra página, por exemplo).

```
<iframe src="pagina_ou_link.html" width="100%" height="500" [frameborder="0"]></iframe>
```

## 8. Flexbox (ou Display Flex):

Estruturação de layouts.



Composto por um elemento pai (<DIV>, <SECTION>, <ARTICLE>, etc.) e elementos filhos (<DIV> interna, <p>, <h1> e etc). O container será transformado em FLEXBOX e irá determinar o alinhamento e a formatação dos filhos.

#### - Propriedades aplicadas no elemento pai:

Display: flex; //transforma o container em elemento flexbox

Flex-direction: row, column, row-reverse, column-reverse. Inverte os eixos. Por padrão, a flexdirection é row, e o eixo principal é o X da horizontal (X). Ao utilizar flex-direction: column, há uma inversão dos eixos, passando o eixo principal a ser o Y (horizontal) e X o eixo transversal.

Por padrão, a altura/largura dos flex itens (elementos filhos) são limitados pelo tamanho da Flex. Pode-se utilizar a propriedade flex-wrap: wrap; no Flex para fazer com que os elementos quebrem a linha, caso a dimensão dos itens extrapole a dimensão do pai.

Flex-flow: <direção> <quebra>;

Flex-flow: row wrap;

Justify-content: alinha os elementos filhos no eixo principal (por padrão, na horizontal, com flexdirection: row). Centraliza, à esquerda, à direita, space-between e space-around.

Align-items: alinha os elementos filhos no eixo transversal (por padrão, na vertical, com flexdirection: row). Centraliza, acima, abaixo, space-between e space-around. Baseline alinha à base da maior fonte.

#### - Propriedades aplicadas nos elementos filhos:

Flex-basis: define o tamanho do elemento, conforme o eixo. Se flex-direction: row (eixo principal horizontal), definirá a largura do objeto. Se flex-direction: column; definirá a altura do objeto.

Flex-grow: pega o espaço sobrando entre os elementos, e aplica ao elemento selecionado. Pode se ter mais de um flex-grow, um para mais cada elemento. Ocorre quando se aumenta o tamanho da tela.

```
.flexbox div:nth-child(2) {
    Flex-grow: 2;
}
```

```
.flexbox div:nth-child(4) {
    Flex-grow: 1;
}
```

Flex-shrink: distribui o espaço que excede. Ocorre quando se diminui a tela, de forma que os elementos não cabem. O elemento selecionado com *flex-shrink* será diminuído.

```
.flexbox div:nth-child(2) {
    Flex-shrink: 2;
}
```

Flex: atalho para flex-grow, flex-shrink e flex-basis, nessa ordem.

Order: Define a ordem dos elementos dentro de uma Flexbox. Quando os demais elementos não possuem ordem, qualquer ordenação leva o elemento para o fim. Assim, se atribuir uma ordem com valor negativo, o elemento é levado ao início.

Align-self: alinha verticalmente o elemento que recebe a propriedade, independentemente do alinhamento dos demais filhos.

### Dicas no exercício:

Fazendo os cards com mais facilidade: Ao aplicar o **display: flex;** no container, as <DIV> interna (cards) já se alinham automaticamente, e sua altura obedece à altura do container (e não do conteúdo de cada card). Pode ser utilizado em substituição ao **float: left;** ou display: **inline-block;** que seria implementada em cada elemento filho.

Alinhando os elementos internos de cada card: transformar cada card (seja ele <DIV> ou <article> ou qualquer container) também como **display: flex;** Depois, alterar o eixo para a horizontal com **flex-direction: column;**

Exemplo:

```

<style>
  .cardContainer{
    display: flex;
  }

  .cardFlex{
    border: 1px solid black;
    width: 30%;
    padding: 10px;
    margin: 1%;
    display: flex;
    flex-direction: column;
  }

  .cardFlex p{
    flex-grow: 1;
  }

  .cardFlex button{
    align-self: flex-end;
  }
</style>
</head>
<body>
  <h2>Display Flex</h2>
  <p>Acomodar ou distribuir elementos de modo fácil</p>
  <p>Ajuda a centralizar na vertical</p>

  <section class="cardContainer">
    <article class="cardFlex">
      <h3>titulo</h3>
      <p>scxzc xv xv c zc xcvxcv,m ncbn çcklvbncvç lbknvbkļ lk
cvn blçkcvb nçvklbj </p>
      <button>Vai</button>
    </article>

    <article class="cardFlex">
      <h3>titulo</h3>
      <p>scxzc xv xv c asda cvvxxcv cc bv vn bn scxzc xv xv c a
sda cvvxxcv cc bv vn bn bn scxzc xv xv c asda cvvxxcv cc bv vn bn</p>

      <button>Vai</button>
    </article>

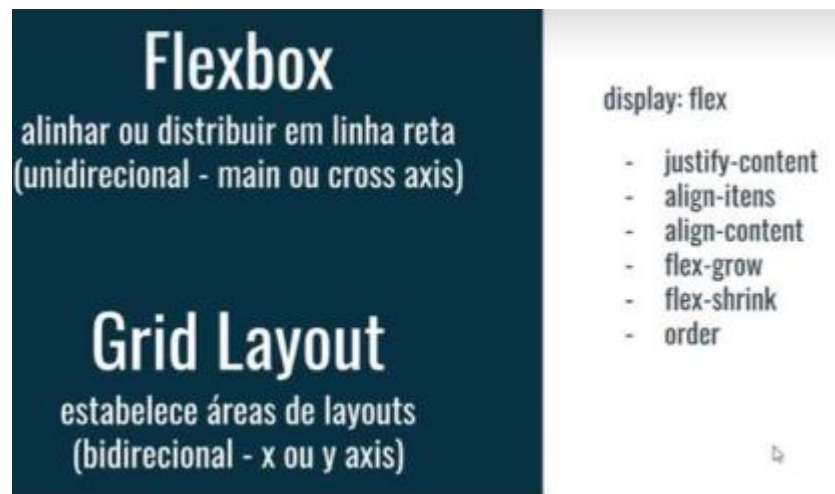
    <article class="cardFlex">
      <h3>titulo</h3>
      <p>scxzc xv xv c vbnvbnvbn kzxcłk</p>
      <button>Vai</button>
    </article>
  </section>
</body>

```

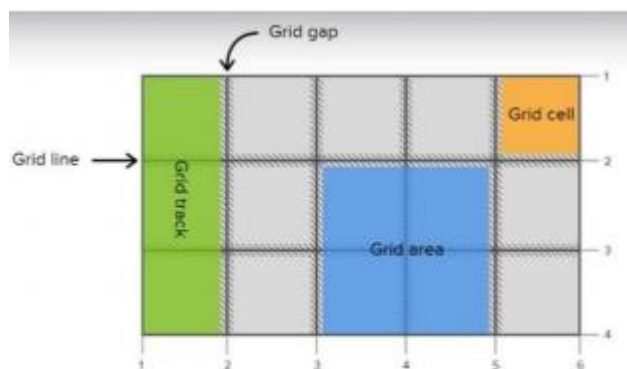


## 9. Grid Layout (ou Display Grid ou CSS Grid):

Diferença entre Flexbox e Grid Layout:



Grid layout estabelece áreas, como header, footer e etc.



- Display: grid; Transforma o container em uma Flex Grid.

- **grid-template-columns:** [auto] <tamanho coluna1> <tamanho coluna2> <tamanho coluna3> ;

Define a quantidade de colunas que conterà o grid. Exemplo:

```
grid-template-columns: 33.33% 33.33% 33.33%; /* Define 3 colunas com 33%
cada */
```

- **grid-template-rows:** [auto] <tamanho linha1> <tamanho linha2> <tamanho linha3> ;

Define a quantidade de linhas que conterà o grid. Exemplo:

```
grid-template-rows: 50% 50%; /* Define 2 linhas com 50% de altura cada */
```

**Função Repeat:** repete um determinado valor. Exemplos:

```
grid-template-columns: repeat(3, 100px); /*define 3 colunas com 100px
cada */
```

```
grid-template-columns: repeat(2, 100px) 90px; /* define 3 colunas,
sendo as duas primeiras com 100px cada, e a Terceira com 90px. */
```

**Atributo Auto:** Assume a largura e a altura necessária para caber o conteúdo, respeitando o padding e margin.

**Unidade FRAÇÃO:** Especificamente para Flex Grids, define em frações o tamanho de cada elemento do grid. Distribui o espaço interno igualmente para os elementos. Exemplos:

```
grid-template-columns: repeat(3, 1fr); /* define 3 colunas, cada uma
ocupando 1 fração do tamanho total, ou seja, cada uma delas ocupando 1/3
do container */
```

```
grid-template-columns: 100px repeat(2, 1fr); /* define 3 colunas, sendo a
primeira fixa com 100px e as outras duas iguais, cotendo o espaço que
sobrou dentro da container. Se o container tiver largura em porcentagem,
as duas últimas colunas, que recebem a fração, irão variar de tamanho à
medida que a tela for aumentada ou diminuída, enquanto a primeira
permanecerá fixa com 100px. A segunda e a terceira coluna também poderiam
ter valores fracionados distintos */
```

- **grid-row-gap ou grid-column-gap:** define uma margem entre as linhas (row-gap) ou colunas (column-gap) dos elementos. Exemplo:

```
grid-row-gap: 10px;
grid-column-gap: 10px;
```

- **grid-gap: row column:** Atalho para as margens internas, em substituição à grid-row-gap e gridcolumn-gap.

Exemplo:

```
grid-gap: 10px 10px;
```

-Mudando a posição dos elementos na Flex Grid:

**Atenção:** uma Grid com 3 colunas possui 4 gridlines (contando também as bordas). Da mesma maneira, duas linhas possuem 3 gridlines.

grid-row-start: gridline de início da linha;

grid-row-end: gridline de fim da linha;

grid-column-start: gridline de início da coluna;

grid-column-end: gridline de fim da coluna;

Exemplo:

```
.gridItem:nth-child(1){
    background: red;
    grid-row-start: 1;
    grid-row-end: 2;
    grid-column-start: 3;
    grid-column-end: 4;
}
```

Atalhos:

Grid-row: <gridline linha de inicio> / <gridline linha do fim>;

Grid-column: <gridline coluna de inicio> / <gridline coluna do fim>;

Passando o número -1 significa a **última linha**. Útil se quiser fazer mesclagem de itens.

- Mesclagem de elementos: **grid-column / grid-row 2 / span 2;** /\*o elemento iniciado da gridline 2 ocupará o espaço de 2 elementos.

- Criando Template Areas: definindo nomes para as células.

**Grid-template-areas**: "<nomeA1> <nomeB1> <nomeC1>" /\* Para uma Grid de 3 x3  
 "<nomeA2> <nomeB2> <nomeC2>"  
 "<nomeA3> <nomeB3> <nomeC3>"

Etapas:

- a. Criar a container e os elementos (griditems) no HTML. Definir uma classe par ao container e classes para os elementos, conforme agrupamento no layout.
- b. No CSS, definir o container com display: grid; e criar as células e colunas através de gridtemplate-columns e grid-template-rows.
- b. Definir os nomes através de grid-template-areas.
- d. Criar as classes dos elementos, através de grid-area.

Se quiser que um campo fique "em branco", define o nome como "."

Exemplo:

```

<style>
  .gridContainer2{
    width: 300px;
    background-color: cornflowerblue;
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-template-rows: repeat(3, 100px);
    grid-gap: 10px;
    grid-template-areas:
      "header header header"
      "main main nav"
      "main main ."
  }

  .gridItem2{
    border: solid 1px black;
  }

  .gridItem2:nth-child(even){
    background-color: #aaa;
  }

  .gridItem2:nth-child(odd){
    background-color: #eee;
  }

  .header{
    grid-area: header;
  }

  .main{
    grid-area: main;
  }

  .nav{
    grid-area: nav;
  }
</style>

```

No HTML:

```

<body>
  <h2>Grid Layout</h2>

  <div class="gridContainer2">
    <div class="gridItem2 header">A</div>
    <div class="gridItem2 header">B</div>
    <div class="gridItem2 header">C</div>
    <div class="gridItem2 main">D</div>
    <div class="gridItem2 main">E</div>
    <div class="gridItem2 nav">F</div>
    <div class="gridItem2 main">G</div>
    <div class="gridItem2 main">H</div>
    <div class="gridItem2 nav">I</div>
  </div>
</body>

```

- **Gridlines nomeadas:** ver aulas 140 e 141.

- **Auto-fill vs auto-fit e auto-rows:**

Utilizados para que os grid-itens possam se adequar ao tamanho do grid container. Útil para dar certa responsividade. Os itens vão se ajustando em colunas ou linhas, de acordo com o tamanho da tela.

grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));

/\* define n colunas com  
no mínimo 100px de  
largura e no máximo  
1fr. \*/

grid-auto-rows: 100px;

/\*define n linhas, todas  
com 100px de altura.  
Poderia colocar mais de  
um valor. Neste caso, a  
primeira linha teria o  
primeiro valor, a  
segunda linha o  
segundo valor e assim  
sucessivamente. \*/

## 10. MAIS SOBE SEMÂNTICA, ESPECIFICIDADE E DICA DE ALTURA NOS ELEMENTOS:

### ➤ Mais sobre semântica e CSS:



- cite
- blockquote
- address
- code
- figure e figcaption

- `<abbr title="descrição da abreviação"> texto </abbr>`: marca um conteúdo de texto como abreviação. O browser pode sublinhar tracejado.

- `<time datetime="2018-09-15T20:00:00-03:00"> texto </time>`: marca um conteúdo de texto como data. O atributo datetime é formado por PADRÃO[THORA-FUSO HORARIO].

- `<del> texto </del> <ins> texto </ins>`: marca um conteúdo como excluído para a adição de outro texto. Como um changes track.

- `<kbd> texto </kbd>`: marca um conteúdo de texto como teclas. O browser apresenta formatação diferenciada.

- `<span lang="em-us">texto</span>`: define que o determinado texto marcado está em outra língua.

**Especificidade no CSS:** Como o CSS aplica estilo quando há algum conflito entre as regras do CSS. Classes são mais específicas que a formatação de uma TAG. No entanto, a TAG com classe, torna-se mais específica que a classe. Id# são mais específicos que classes.

Menos específico

Mais específico

- \* > P > .classe > p.classe > id# > estilo inline

Calcular a especificidade					
	Estilo inline	Id #	Atributos e pseudoclasses	Tags e pseudoelementos	Resultado
*	0	0	0	0	0000
p	0	0	0	1	0001
.destaque	0	0	1	0	0010
p.destaque	0	0	1	1	0011
p[lang]	0	0	1	1	0011
#unico	0	1	0	0	0100
#nav ul li a	0	1	0	3	0103
p:first-child	0	0	1	1	0011
p:first-letter	0	0	0	2	0002

Diretiva **important** > Atribui especificidade máxima. Exemplo:

```
P {
    Color: green !important;      /* todos os <p> ficarão verdes, independentemente de
                                   existirem classes ou #id ou inline ou etc. */
}
```

### ➤ Altura de 100% no CSS (definindo altura em porcentagem):

Para que a altura de um elemento funcione com sua definição em porcentagem, é necessário que o container tenha anteriormente a definição de altura.

Para que altura do body possa funcionar em porcentagem, até a TAG <HTML> precisa estar definida com height: 100%; **TODOS** os elementos pai precisam estar com a altura definida.

Para resolver a questão de forma mais simples, pode-se utilizar a UNIDADE DE MEDIDA vh (viewport height). Exemplo:

```
div {
    height: 90vh;      /* altura de 90% da Viewport */
}
```

Assim como existe a unidade de medida vh (viewport height), existe também a vw (viewport width).

## 11. Animação

### Revisar Transições

## Relembrar

transition

- transition-property
- transition-duration
- transition-delay
- transition-timing-function

## Transition (shorthand)

```
transition: property duration timing-function delay;
```

## Múltiplas transições

```
transition: property1 duration1 timing-function1 delay1,
            property2 duration2 timing-function2 delay2;

transition-property:      property1,    property2;
transition-duration:     duration1,    duration2;
transition-timing-function: func1,     func2;
transition-delay:        delay1,       delay2;
```

➤ Animation & Keyframes:

Keyframe: cria animação.

Animation: aplica a animação no HTML.

Transition vs animation:

**Transition:** apenas transições simples, sem intermediários.

depende de uma ação do usuário (:hover, :focus, :active e etc.)

**Animation:** animações mais complexas, com frames.

Não dependem da interação do usuário.

Maior controle sobre a animação criada.

Os keyframes devem ser determinados em porcentagem, dentro da timeline.

Apenas valores numéricos podem ser animados.

Propriedades mais performáticas: transform e opacity.

Propriedade **will-change**: informa que alguma propriedade sofrerá uma animação.

Preparação do browser para melhorar a performance. Não se pode utilizar em todos os elementos, de outra maneira prejudica o desempenho.

**Declarando os Keyframes:**

```
@keyframes <nome_da_animacao>{
    0% {
        /* o mesmo que from */
        Color: red;
    }
    50% {
        Color: blue;
    }
    100% {
        /* o mesmo que to */
        Color: orange;
    }
}
```



Aplicando os keyframes (shorthand):

```
.elemento {
    animation: <nome_da_animacao> <tempo_de_execucao>;
}
```

Outras propriedades (não shorthand):



Ordem correta para aplicação das propriedades em shorthand:

animation: name duration timing-function delay iteration-count direction fill-mode playstate;

Exemplo:

No HTML:

```
<div id="anima-time">
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <div>4</div>
    <div>5</div>
</div>
```

NO CSS:

```
<style>
@keyframes anda{
0%{
transform: translateX(0);
background: yellow;
}
100%{
transform: translateX(300px);
background: blue;
}
}

#anima-time div{
width: 80px;
height: 50px;
border: 2px solid red;
}

#anima-time div:nth-child(1){
/* animation-name: anda;
animation-duration: 2s;
animation-timing-function: ease-in-out; */
animation: anda 2s ease-in-out;
}

#anima-time div:nth-child(2){
/* animation-name: anda;
animation-duration: 2s;
animation-iterationcount: 4; quantidade de repetições. Pode ser infinite
animationdirection: alternate; faz a animação retornar suave */
animation: anda 2s 4 alternate;
}
```

```

#anima-time div:nth-child(3){
/* animation-name: anda;
animation-duration: 2s;
animation-delay: 3s;
animation-iteration-count: infinite;
animation-direction: alternate; */
animation: anda 2s 3s infinite alternate;
}
#anima-time div:nth-child(4){
background: red;
animation-name: anda;
animation-duration: 2s;
animation-delay: 3s;
animation-iteration-count: 2;
animation-direction: alternate;
animation-fill-mode: forwards;
/*backwards: já inicia a animação com as propriedades de 0%,
independentemente do definido no objeto */
/*forwards: quando termina a animação, fixa as propriedades d
e 100%, independentemente do definido no objeto */
/*both: ambos*/
/* animation: anda 2s 3s infinite alternate; */
}
#anima-time div:nth-child(5){
background: red;
animation-name: anda;
animation-duration: 2s;
animation-iteration-count: infinite;
animation-direction: alternate;
animation-fill-mode: both;
animation-play-state: running;
}
#anima-time div:nth-child(5):hover{
animation-play-state: paused;
}
</style>

```

## Parte II – Javascript

### 12. Introdução ao Javascript

Baseado em ECMAScript (linguagem base para outras linguagens, como Javascript, JScript, ActionScript).

Linguagem Single Thread: Monotarefa. O código principal e os temporizadores são executados um de cada vez.

Sintaxe da declaração: Código digitado entre as TAGs **<script> ... </script>**. Recomendável que se implemente o Javascript sempre na última parte da página, imediatamente antes de **</body>**, quanto todos os elementos HTML e CSS já estiverem lidos (interpretados pelo browser). Podese também incluir um arquivo externo com o script, através de **<script src="nome\_do\_arquivo.js"></script>**.

Atenção: JAVASCRIPT É LINGUAGEM CASE SENSITIVE.

Em JavaScript não é necessário terminar as instruções com ; mas é uma boa prática. Utilizando se ponto e vírgula, pode-se declarar mais de uma instrução na mesma linha.

Strings podem ser escritas entre aspas simples ou duplas. Se quiser colocar um texto a ser exibido entre aspas, pode-se colocar a string em aspas simples, e em aspas duplas o texto que se quer colocar entre aspas. Outra maneira, é utilizar o \ (escape) antes das aspas que se quer exibir. Exemplo: var frase = "O \"HTML\" é maneiro!";

No Chrome, pode-se utilizar a guia Console do Inspetor de Elementos para testar o código.

Comentários: // (única linha) ou /\* ... \*/ (múltiplas linhas).

Variáveis sem valores: **null**, definido pelo usuário. **Undefined**, quando não se é atribuído nenhum valor.

Variáveis com valores genéricos (objetos nativos do JS): **NaN**, Not a Number, indica que a variável não possui valor numérico. **Infinity**, quando a variável possui um valor muito alto, tendente ao infinito (**2e308**, 2 elevado a 308 dezenas).

#### Objetos Globais:

window/global (no Node JS): janela do browser. A função **alert()**, por exemplo, é na verdade um método do objeto window (**window.alert()**).

This: representa um objeto dentro de um determinado escopo.

Protótipos: Espécie de “fábrica” de objetos. Meio de herança.

### ➤ Depuração de Código:

CRTL + SHIFT + I no Chrome (ou botão direito e “Inspecionar elementos”).

Console: debugger em linha de comando.

```
Console.log("mensagem")
```

Source: exibe o código fonte de algum arquivo aberto no browser. Pode-se incluir breakpoints e executar o código passo-a-passo (tecla F9), consultando valores através de hint do mouse ou através do painel Console. Via código, pode-se incluir a função **debugger**; . Com ela, o browser fará uma parada para execução passo-a-passo e acompanhamento em tempo de execução.

### ➤ Integração HTML e JavaScript (mais detalhes no tópico “DOM”):

Na TAG HTML, definir uma id para o elemento.

Na seção <script></script>, utilizar **document.querySelector("#<id>");** . Ao atribuir em uma variável, será armazenada como uma instância. Esta instância possui o método **innerHTML**, para incluir conteúdo no HTML. Exemplo:

```
<div id="testeJS"></div>

<script>
    var divHtml = document.querySelector("#testeJS");
    //var output = document.querySelector("#output").textContent;
    //recuperaria o texto de uma <div> com id "output"

    divHtml.innerHTML = "inclusão de texto via JavaScript";
</script>
```

## 13. Algumas Funções, Objetos, Propriedades e Métodos Importantes

Mais detalhes nos tópicos seguintes.

**alert("mensagem")**: Emite alerta na tela.

**prompt("mensagem")**: Caixa de diálogo para informação do usuário alerta na tela. O Script SEMPRE receberá o valor como string.

**confirm("mensagem")**: Função que emite uma alerta para confirmação. Retorna true ou false.

**Typeof <variável>**: Retorna o tipo de uma variável.

**Math.random()**: Retorna um número aleatório entre 0 e 0.99999999999999. Pode-se fazer operações, como **Math.random() \* 2** (útil para capturar números acima de 1).

**document.write("texto")**: escrever na tela (pouco profissional).

**console.log()**: Visualiza o conteúdo no Console do Chrome. Estilo de um debug.print do VBA.

**IsNan(<parâmetro>)**: Is Not a Number. Se puder ser convertido em número, retorna FALSE. Se

o parâmetro não puder ser convertido para número, retorna TRUE.

String **'use strict'**; implementada no início de uma função (primeira instrução). Exige que todas as variáveis sejam declaradas.

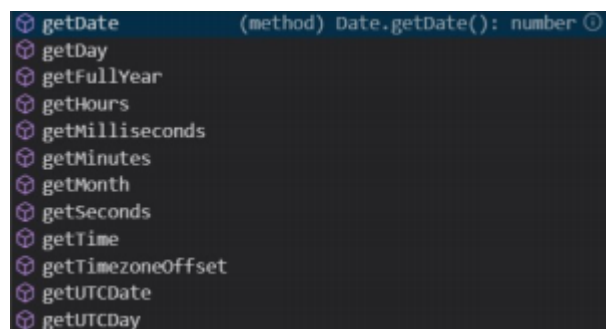
**throw new Error ("Mensagem de Erro")**: Exibe "Mensagem de Erro" no console do browser, e não executa o restante do comando.

**debugger**; Faz uma parada na execução do código, para inspeção.

**date()**: Objeto que retorna a data atual. Deve ser atribuída a uma variável

Sintaxe: **var<variável> = new Date()**

O objeto <variável> passará a ter alguns métodos, tais como:



## 14. Sintaxe e Lógica de Programação em Javascript

### 14.1 Variáveis:

Definição:

Var <nome\_da\_variavel> = valor;

OU

let <nome\_da\_variavel> = valor;

A diferença entre **var** e **let** diz respeito ao escopo.

Uma boa prática é nomear variáveis que representam instâncias do Dom iniciando com \$.

### 14.2 Constantes:

Const <nome\_constante> = valor;

### 14.3 Conversão de Dados:

**parseInt(<valor>)**: Converte para inteiro.

**parseFloat(<valor>)**: Converte para número real.

## 14.4 Operadores:

### Atribuição:=

**+=** soma e atribuição simultânea.

**-=** subtração e atribuição simultânea.

**\*=** multiplicação e atribuição simultânea.

**/=** divisão e atribuição simultânea.

### Concatenação: +

Conversão em booleano: !! (**null** retorna **false**, qualquer valor retorna **true**)

### Aritméticos:

Adição: +

Subtração: -

Multiplicação: \*

Divisão: /

Módulo (resto da divisão): %

Exponenciação: \*\* (ou e. Exemplo: 2e3 é o mesmo que  $2 * 10^{**3}$ ).

Incremento: <variável>++ ou ++<variável>

Decremento: <variável>-- ou --<variável>

### Comparação:

Igual (valor): == // '2' == 2 retornará true, pois apenas os valores são comparados.

Diferente (valor): !=

Igual (valor e tipo): === // '2' === 2 retornará false, pois valor e tipo são comparados.

Diferente (valor e tipo): !==

Maior que: >

Menor que: <

Maior ou igual: >=

Menor ou igual: <=

### Lógicos (em ordem de precedência):

Não (inversão): ! // x=2 e y=3. !(x>y) retorna true.

E: &&

Ou: ||

## 14.5 Estruturas de Controle:

### - Condicionais:

If...else:

If<condição>{

....

}

Else if {

...

}

Else {

...

}

Exemplo:

```
<script>
    var n1 = parseFloat (prompt("Digite o primeiro número:"));
    var n2 = parseFloat (prompt("Digite o segundo número:"));

    var media = (n1+n2)/2;

    if (media >= 5) {
        alert("Média do aluno: " + media + ". APROVADO!");
    }
    else {
        alert("Média do aluno: " + media + ". Reprovado!");
    }
</script>
```

Operador ternário, em substituição ao if:

<expressão> ? <valor/operações true> : <valor/operações false>;

### Switch ... case:

Switch(<elemento a verificar>){

Case <condição1>:

...

Break;

Case <condição2>:

...

Break;

Case <condição3>:

...

Break;

Default:

...

}

&gt; Repetições:

- While:

While (&lt;condição&gt;) {

...

}

- do while:

Do {

...

} while(&lt;condição&gt;)

- for:

For(&lt;condição inicial&gt;;&lt;condição final&gt;;&lt;incremento&gt;){

...

}

Exemplo:

```
var i = 1;

for (i=1;i<=1000;i=i+1){
    document.write("Imprimindo o número " + i + "<br>")
}
```

## 14.6 Funções:

### ➤ Declarações, chamadas e argumentos.

Executar uma função quando se pressiona um botão, por exemplo: Atributo **onClick="função()"** num elemento **<button>**. Método em desuso, como será visto nos tópicos adiante.

Definição no **<script></script>**:

Function <nome da função>([parâmetros]){

...

[return <resultado>;]

}



Exemplo:

```
<body>
  <h1>Exercício Javascript - Área de um Retângulo</h1>

  <button onclick="CalcArea(vAltura, vLargura)">Calcular área do Retângulo</button>

  <script>
    var vAltura = parseFloat(prompt("Informe a Altura:"));
    var vLargura = parseFloat(prompt("Informe a Largura:"));

    function CalcArea(Alt, Larg) {
      alert("Área: " + (Alt * Larg));
    }

  </script>
</body>
```

Chamar a função:

<nome\_da\_função>();

Ou

<nome\_da\_função>.call();

Porém, se houver parâmetros:

<nome\_da\_função>.call(this, <parametro1>, <parametro1>, ...);

Ou

<nome\_da\_função>.apply(this, [<parametro1>, <parametro1>, ...]); //array de  
//parâmetros

Sempre que houver *return*, sai da função e o restante do bloco não será executado. Não é obrigatório que uma função retorne valor, logo, não há obrigatoriedade da instrução *return*.

Função Anônima: Em alguns casos, o nome da função não é obrigatório, podendo ser atribuída a uma variável, que se transformará em um método. Atenção à obrigatoriedade do ; fechando a função. Exemplo:

```
var teste = function (str) {alert(str);}; //definição do método
```

```
teste("Mensagem de Teste"); //chamada do método
```

Funções Void: o mesmo que Procedimentos, ou seja, funções que não retornam valor.

Na chamada das funções, os parâmetros são passados entre vírgulas.

São objeto de primeira classe, portanto podem ser tratadas como qualquer objeto. Pode

inclusive receber uma propriedade. Por exemplo, se existir uma function Somar (x,y) {return x+y;}, esta função pode receber uma propriedade do tipo Somar.descricao = “função somar”; cuja propriedade pode ser acionada normalmente. A própria função pode ser passada como parâmetro de outra função (funções recursivas ou callbacks).

Propriedade name: Retorna o nome de uma função.

Apenas as funções criam escopo próprio para as variáveis. Uma variável definida dentro de uma função terá “validade” apenas dentro da função, salvo se for criada SEM a instrução **var** (quando aí passará a ter escopo global).

Uma boa prática para evitar variáveis globais, que é o padrão, é implementar todo o código dentro de uma função autoinvocável (IIFE).

IIFE (Immediately-Invoked Function expression): Funções auto-executáveis.

```
(function() {  
    ...  
})();
```

Boa prática utilizar a string **‘use strict’**; como primeira instrução da função, para obrigar declaração de variáveis.

Método **arguments**. Implementado DENTRO da função, retorna quantos e quais os argumentos que foram recebidos.

Acessando um parâmetro (argumento) específico: **arguments[índice]**.

Quantidade de argumentos passados: **arguments.length**.

Funções com quantidade indefinida de argumentos: Utilizar arrays. Na declaração, definir apenas 1 argumento, que receberá um array. Na chamada, passar a quantidade desejada de argumentos.

Exemplo:

```
(function (){
    function Somar(){
        'use strict';

        var i=0;
        var Soma=0;

        console.log(arguments);

        while (typeof arguments[i] == "number")
            Soma += arguments[i];
            i++;
        }

        //Outra forma de fazer. Porém, seria necessário
        //definir o parâmetro arr.
        // for (i=0; i < arr.length; i++){
        //     Soma += arr[i];
        // }

        console.log(Soma);
    }

    Somar(10, 5, 20);
})();
```

**>Arrow function:** Introduzido com o ECMAScript 2016. Trata-se de uma notação diferenciada para declaração de funções, especialmente quando é uma função que funciona como método de um objeto. Exclui-se a palavra reservada **function** e inclui-se os caracteres =>

Exemplo:

<p><i>Função normal (função anônima):</i></p> <pre>function(nota) {     return nota === 0; }</pre>	<p><i>Versão Arrow function:</i></p> <pre>(nota) =&gt; {     return nota === 0; }</pre> <p>Se tiver apenas um parâmetro, ainda se pode eliminar os parênteses:</p> <pre>nota =&gt; {     return nota === 0; }</pre> <p>Se tiver apenas uma única linha, pode-se eliminar as chaves e a palavra <b>return</b>:</p> <pre>nota =&gt; nota === 0;</pre>
--	---

### 14.7 Tratamento de Erros:

Utiliza `throw new Error("mensagem")` dentro da função que vai ser testada. Posteriormente, utiliza o `try{...} catch(e){...}` para testar se essa função que contém o `throw` resultou em algum erro.

```
Try{
    .... //função a ser testada, que contém o throw new Error
}
catch(e) {
    console.log(e); //tratamento do erro, se houver
}
```

Exemplo:

```
function MostrarAreaOutput() {
    var vAltura = parseFloat(prompt("Informe a Altura:"));
    var vLargura = parseFloat(prompt("Informe a Largura:"));
    TrataErroNumber(vAltura, vLargura);
    output.innerHTML = "Área do retângulo: " + CalcArea(vAltura, vLargura);
}

function CalcArea(a, l){
    if (isNaN(a) || isNaN(l)) {throw new Error("Não eh numero válido!");}
    return a * l;
}

function TrataErroNumber(x, y){
    console.log("testando");

    try {
        CalcArea(x, y);
    }
    catch(e) {
        //console.log(e);
        alert(e);
    }
}
```

### 14.8 Arrays:

`var <nome_do_array> = new Array(qtd. elementos);`

ou

`var <nome_do_array> = new Array(<elemento1>, <elemento2>, <elemento3>, ...);`

ou

```
var <nome_do_array> = [<elemento1>, <elemento2>, <elemento3>, ...];
```

- Acessando os elementos:

```
<nome_do_array>[índice];
```

- Adicionando itens, sem informar o índice: método **push**:

```
<nome_do_array>.push(<valor1>, <valor2>, <valor3>, ...);
```

- Convertendo o conjunto array para string: método **toString**.

```
<nome_do_array>.toString(); //Retorna uma string contendo todos os elementos.
```

- Tamanho do array (quantidade de itens): Método **length**.

```
<nome_do_array>.length;
```

Exemplo:

```
<div id="output"></div>

<script>
    var output = document.querySelector("#output");
    var pessoas = new Array();
    var i;
    var msg="";

    pessoas[0]="Helder";
    pessoas[1]="Jamir";
    pessoas[2]="Socorro";

    pessoas.push("Helber");

    for (i=0;i<=pessoas.length-1;i++){
        msg += "Pessoa " + (i+1) + ": " + pessoas[i] + "<br>";
    }

    output.innerHTML = msg;
```

## 15. Objetos em Javascript:

Conjunto de dados agrupados entre {}.

```
var <nome_do_objeto> = new Object();
```

ou

```
var <nome_do_objeto> = {'propriedade1': 'valor1', 'propriedade1': 'valor1',...}
```

- Propriedades do Objeto (variáveis atreladas aos objetos):

```
<nome_do_objeto>.<propriedade_desejada> = valor;
```

- Métodos dos objetos (funções atreladas aos objetos):

```
<nome_do_objeto>.<nome_do_método> = function([parâmetros]) {
  ...
}
```

Exemplo:

```
var produto1 = new Object();
var msg;

//Propriedades
produto1.Nome = "Caneta";
produto1.Preco = 5;
produto1.Estoque = 15;

//Métodos
produto1.Comprar = function(){
    produto1.Estoque--;
}

produto1.Comprar();
```

Outra maneira de criar objetos, com propriedades e métodos:

```
var caneta1 = {
  cor: 'red',
  mudaCor: function(cor) {
    this.cor = cor;
  }
}
```

- **Arrays de Objetos:** Conjunto de dados agrupados dentro de um array.

```
var output = document.querySelector("#output");
var produtos = [
  {"nome": "caneta", "preco": 2},
  {"nome": "lapis", "preco": 1.2},
  {"nome": "borracha", "preco": 0.5}
];

var msg="";
var i=0;

for (i=0;i<produtos.length;i++){
  msg += "Produto: " + produtos[i].nome + "<br>" +
        "Preço: " + produtos[i].preco + "<hr>"
}

output.innerHTML = msg;
```

## 16. DOM (Document Objective Model): Integração HTML e Javascript

DOM = objeto document.

Dica: Acessar várias variáveis com nomes parecidos.

```
var pessoa1;
var pessoa2;
var pessoa3;
var i;

for (i=1; i <= 3; i++){
    window["pessoa" + i] = "texto inserido" + i;
    console.log(window["pessoa" + i]);
}
```

### 16.1 Atributos e Métodos do DOM (Objeto Document):

- **getElementById**: Cria uma instância de um elemento específico HTML, selecionando pelo id.

Sintaxe: `document.getElementById("id_do_elemento");`

- **getElementsByClassName**: Cria uma instância de uma coleção de elementos (Node List) HTML, selecionando pela classe.

Sintaxe: `document.getElementsByClassName("classe_do_elemento");`

Ao acessar a instância, SEMPRE será necessário identificar o index, já que retorna uma coleção de objetos, uma vez que podem existir muitos objetos com a mesma classe.

- **getElementsByTagName**: Cria uma instância de uma coleção de elementos HTML (Node List), selecionando pela TAG.

Sintaxe: `document.getElementsByTagName("TAG_que_deseja_recuperar");`

Ao acessar a instância, SEMPRE será necessário identificar o index, já que retorna uma coleção de objetos, uma vez que irá retornar TODOS os elementos com uma TAG (<p>, <h1>, por exemplo).

- **querySelector**: Semelhante aos anteriores, selecionando APENAS UM elemento (se houver mais de um, será selecionado apenas o primeiro. Se usar índices, dá erro). No entanto, é necessário informar os seletores complexos do CSS. Se for selecionar por id, por exemplo, é necessário informar o #. Pode-se inclusive selecionar elementos descendentes, no mesmo padrão do CSS.

Sintaxe: `document.querySelector("<seletor><nome_do_objeto> <elemento/TAG filho>");`

Exemplos:

```
document.querySelector("#idmain");
```

```
document.querySelector("#idmain p"); //retorna os parágrafos filhos de #idmain
```

- **querySelectorAll**: Semelhante aos anteriores, selecionando UMA COLEÇÃO de elementos (logo, será necessário informar o índice). No entanto, é necessário informar os seletores complexos do CSS. Se for selecionar por classe, por exemplo, é necessário informar o . Pode-se inclusive selecionar elementos descendentes, no mesmo padrão do CSS.

Sintaxe: `document.querySelector("<seletor><nome_do_objeto> <elemento/TAG filho>");`

Exemplo:

```
document.querySelectorAll(".csection p");
```

- **createElement**: Cria um elemento no DOM, atribuindo a uma variável (instância).

Sintaxe: `var <nome> = document.createElement(<tipo de elemento (TAG HTML)>);`

Exemplo: incluindo uma <li> em uma <ul>:

```
function addTask(){
    var $ul = document.querySelector("ul");
    var $txtTask = document.querySelector("#txtTask");
    var li = document.createElement("li");
    var text = document.createTextNode($txtTask.value);

    li.appendChild(text);
    $ul.appendChild(li);
}
```

### - Atributos e métodos das instâncias do DOM:

**textContent**: define ou recupera um texto puro para o objeto. Não reconhece TAGS HTML, por exemplo.

**innerHTML**: define ou recupera um texto do objeto, reconhecendo as TAGS HTML.

Para receber/setar valores de formulário, sempre acessar a propriedade VALUE da instância.

**<instância>.setAttribute("<propriedade>", "valor")**: Atribui ao objeto, via instância, um determinado valor a uma determinada propriedade.

Exemplos:

```
$txtMsg.setAttribute("maxlength", "150");
```

(sendo \$txtMsg uma instância de <textarea>.)



```
$txtMsg.setAttribute("class", "classe");
```

**<variável> = <instância>.getAttribute("<propriedade>"):** Atribui a uma variável o valor de uma determinada propriedade.

**style:** Atribui um CSS inline a um elemento HTML.

Sintaxe: <instância>.style.<propriedade\_CSS> = "<valor>";

Exemplo:

No HTML:

```
<div id="msgContainer" style="display: none;">Restam <span id="resta"></span> caracteres.</div>
```

No JS:

```
var $msgContainer = document.getElementById("msgContainer");
$msgContainer.style.display = "block";
```

**classList.add("nome da classe") e/ou classList.remove("nome da classe"):** Inclui ou remove uma classe ao objeto. Atentar ao suporte! (<https://caniuse.com/>).

**classList.toggle("nome da classe"):** Se o elemento já contém class="nome da classe" este é removido. Se não possuir, é acrescentado. Atentar ao suporte! (<https://caniuse.com/>).

**className:** Insere uma classe no elemento selecionado. Sintaxe:

<instância>.className = "<nome\_da\_classe\_a\_inserir>";

**focus():** Coloca o cursos em um elemento de formulário

.

**value:** Atributo. Para acessar campos de formulário, é necessário utilizar o atributo value.

Quando for um atributo que não recebe valores no HTML (tal como **disabled** ou **readOnly**), pode ainda ser atribuído valor **true** ou **false**.

**keyCode:** Retorna em ASC a tecla pressionada em um campo de texto. Comumente associada a um evento keyup ou input de objetos de texto.

**Atenção 1:** tudo que vem da interface do usuário (ou do DOM) é uma string.

**Atenção 2:** Algumas propriedades tem grafia diferente entre o HTML e o JS. São elas:

atributo no HTML		propriedade no Javascript
for	/	forhtml
class	/	className
readonly	/	readOnly
maxlength	/	maxLength
cellspacing	/	cellSpacing
rowspan	/	rowSpan
colspan	/	colSpan
tabindex	/	tabIndex
cellpadding	/	cellPadding
contenteditable	/	contentEditable
frameborder	/	frameBorder
usemap	/	useMap

Exemplo:

```
<body>
  <h1>Curso Javascript Completo 2018</h1>

  <main class="cmain" id="idmain">
    <h2>subtitulo</h2>
    <p id="pmain" class="pmain">Lorem ipsum dolor sit amet, conse
ctetur adipiscing elit. </p>
  </main>

  <section class="csection" id="idsection">
    <h3>subtitulo</h3>
    <p class="paragrafos">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. </p>
    <p class="paragrafos">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. </p>
    <h3>subtitulo</h3>
    <p class="paragrafos">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. </p>
    <p class="paragrafos">Lorem ipsum dolor sit amet, consectetur
adipiscing elit. </p>
  </section>

  <script>
    (function(){
      'use strict';
      var Texto = prompt("Digite um texto qualquer para o main:
");
```

```

        var pMain = document.getElementById("pmain");
        var pSection = document.getElementsByClassName("paragrafo
s");

        var ObjTAG = document.getElementsByTagName("h3");
        var ObjQuerySel = document.querySelector("#idsection p");
        var ObjQuerySelAll = document.querySelectorAll(".csection
p");

        pMain.textContent = Texto; //pelo id
        pSection[0].textContent = "Mesmo texto, mas para o parágr
afo index 0 da section: <br>" + Texto; //pela classe
        ObjTAG[0].textContent = "Isto é um subtítulo \<h3\>"; //p
ela TAG

        alert(ObjQuerySel.textContent); //QuerySelector
        ObjQuerySelAll[3].innerHTML = "Teste de <b>querySelectorA
ll</b>";

        })();
    </script>

</body>

```

➤ **Capturando o valor de um <input type = “text”>:**

- Definir um value para o <input> no HTML.
- Criar uma instância no <script>.
- Recuperar a propriedade value.

```

<body>
  <h1>Curso Javascript Completo 2018</h1>

  <label>
    Digite seu nome:
    <input type="text" id="txt_Nome">
  </label>
  <button id="botao" onclick="EscreveMsg()">Apenas clique</button>
  <div id="PainelMsg"></div>

  <script>
    var PainelMsg = document.querySelector("#PainelMsg");

    PainelMsg.innerHTML = "Bem vindo, ";

    // (function(){
    function EscreveMsg(){
      var Nome = document.getElementById("txt_Nome");

      PainelMsg.innerHTML += "<b>" + Nome.value + "</b>.";

    }
    // })();
  </script>

</body>

```

➤ **Trabalhando com elementos filhos:**

`<instância>.querySelectorAll("<elemento>")`

Exemplo:

`var Linhas = document.querySelectorAll("tr");` //Linhas recebe todas as `<tr>`

`var Colunas = Linhas[0].querySelectorAll("td");` //Colunas recebe todas as colunas da  
//linha índice 0.

OU

Quantidade de elementos filhos: `<instância>.childElementCount;`

Acessando o elemento filho: `<instância>.children[índice];`

Exemplo:

```
var i=0;
var x=0;
var NotaAtu = 0;
var PontosTot = 0;
var QtdNotas = 0;
var Aluno = document.getElementsByClassName("aluno");

for (i=0; i < Aluno.length; i++){
    NotaAtu=0;
    PontosTot=0;
    QtdNotas=0;
    for(x=0; x < Aluno[i].childElementCount;x++){
        NotaAtu = parseFloat((Aluno[i].children[x].textCo
ntent));

        if (!isNaN(NotaAtu)){
            PontosTot += NotaAtu;
            QtdNotas++;
        }
    }
    //Final da captura de todos as células de um aluno
    //Aluno[i].children[QtdNotas+1].textContent = PontosT
ot / QtdNotas;
    Aluno[i].querySelector(".aluno-
m").textContent = PontosTot / QtdNotas;
```

**appendChild:** Incluir um elemento filho. Sintaxe: `<elemento_pai>.appendChild(<filho>);`

**removeChild:** excluir um elemento filho. Sintaxe: `<elemento_pai>.appendChild(<filho>);`

## 17. Objetos em Javascript:

Possuem propriedades e métodos. Utilizar o Console do Chrome para orientação.

Observação: Arrays são objetos em JavaScript, logo são **passados por referência** nas funções.

➤ **Construtores:** Objetos criados como instâncias, através da função **new**.

Exemplo: `var DataAtu = new Date();`

**Funções construtoras (Aula 249):** Por convenção, nomeá-las com inicial maiúscula.

Propriedades públicas: acessíveis diretamente, define-se como no exemplo abaixo.

Propriedades privadas: definidas como variáveis, e utiliza-se métodos (funções) internos para acessar ou definir seus valores.

```
function <nome_funcao_construtora>(){
  this.<propriedade1> = valor_default;
  this.<propriedade2> = valor_default;
  ...
    this.<método> = function X([parâmetro]){
      ...;
    }
}
```

Para criar um novo objeto:

```
var <novo_objeto> = new <função_construtora>;
```

Exemplo:

```
(function(){
    'use strict'
    //Função construtora
    function Caneta(desc, cor, preco) {
        //se não for passado nenhum valor
        //na construção do objeto,
        //recebe o valor default
        this.desc = desc || "caneta",

        this.cor = cor || "black",
        this.preco = preco || 1,
        this.mudaCor = function(NovaCor){
            this.cor=NovaCor;
        }
        this.mudaPreco = function(NovoPreco){
            this.preco = NovoPreco;
        }
    }

    var caneta1 = new Caneta();
    // debugger;
})();
```

**Atenção:** Se esquecer de informar o **new** na construção do objeto, as propriedades serão aplicadas a **window**. Para a correção

## 18.Eventos em Javascript:

**Event handle:** é a função a ser executada quando ocorre um evento. Exemplos: click, mouseover, mouseout, keyup, submit,load e etc.

### 18.1 Sintaxe (DOM nível 0):

- No Javascript:

<objeto>.<evento> = <função>;

Function <função> {

...

}

Exemplo:

```
<body>
  <h1>Curso Javascript Completo 2018</h1>

  //Não colocar () -> Duas maneiras no DOM nível 0

  <button id="btn1" onclick="clicar()">botão 1</button>
  <button id="btn2">botão 2</button>

  <script>
    var btn2 = document.getElementById("btn2");

    btn2.onclick = clicar; //Não colocar ()
                          //caso contrário, executa

                          //mesmo sem evento

    function clicar(){
      alert("cliquei");
    }
  </script>
```

## 18.2 Sintaxe (DOM nível 2):

**Adicionar evento no DOM 2:** <instância>.addEventListener ("<evento>", <função>);

```
btn3.addEventListener("click", clicar); //Não colocar ()
```

Com o DOM nível 2, é possível atrelar mais de um evento ao mesmo elemento, o que não é possível no nível 0.

Exemplo:

```
btn3.addEventListener("click", function(){alert("Clicou com o dom level 2");});
btn3.addEventListener("click", function(){alert("Clicou com outro evento");});
```

**Remover evento no DOM 2:** <instância>.removeEventListener ("<evento>", <função>);

### 18.3 Alguns Eventos Comuns (DOM2):

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

**click**: aplicado a qualquer objeto.

**submit**: aplicado a formulários.

**Keydown**: evento de teclado. Quando aperta QUALQUER tecla (acionado ANTES de incluir no objeto texto).

**Keypress**: evento de teclado. Quando aperta qualquer tecla, EXCETO não caracteres (Shift, Fn e Caps Lock e etc.). Acionado antes de incluir no objeto texto.

**Keyup**: quando solta a tecla (acionado APÓS a inclusão do texto no objeto).

Input: mudança de valores em um objeto texto.

**load**: Quando um determinado objeto conclui o carregamento. Comumente aplicado a **window** ou **form**.

Exemplo: executar script apenas quando a página terminar o carregamento:

```
(function (){  
    "use strict";  
    window.addEventListener("load", init);
```

```
    function init(){  
        //...  
    }  
  
})();
```

**18.4 Propagação de Eventos (bubbles)**: capacidade do elemento que sofreu o evento propagar para os elementos superiores (até o <body>).

Se houver uma <DIV> dentro de outra <DIV>, por exemplo, e ambas possui o evento **click**.

Quando clicar na <DIV> interna, será executado o evento **click** de ambas.



**Exemplo:**

```

<div class="container">
  <h1>propagação de eventos</h1>

  <div id="d1" style="background: red; padding: 25px;">
    texto dentro da div
    <spam style="background: yellow;">texto dentro da spam</spam>
  </div>
</div>
<script>

  var $d1 = document.querySelector("#d1");
  var $d1Spam = document.querySelector("#d1 spam");

  $d1.addEventListener("click", function(){console.log("clique no $
d1");});
  $d1Spam.addEventListener("click", function(){console.log("clique
no $d1Spam");});
  document.addEventListener("click", function(){console.log("clique
no document");});

</script>

```

Pode-se inverter a ordem da propagação, utilizando o terceiro parâmetro da **addEventListener**, informando **true**. A propagação pode ser interrompida através do método **stopPropagation**, conforme visto a seguir.

**18.5 O Objeto Event:**

Se na função atrelada ao **addEventListener** for incluída um parâmetro, este parâmetro será transformado em um objeto, e este objeto será o próprio evento, podendo inclusive ser inspecionado no Chrome.

**Exemplo:**

```

$d1Spam.addEventListener("click", function(){console.log("cli
que no $d1Spam");});
document.addEventListener("click", function(evt){
  console.log("clique no document");
  console.log(evt);
  console.log(this);
  console.log(evt.target);
  console.log(evt.currentTarget);
});

```

No caso, se utilizasse **evt.stopPropagation()**, interromperia a propagação de eventos.

## 18.6 DELEGAÇÃO DE EVENTOS:

Atrelar um evento a um elemento mais alto e verificar qual objeto que disparou o evento.

Exemplo:

```
<div class="container">
  <h1>Delegação de Eventos</h1>
  <ul>
    <li>lista 1</li>
    <li>lista 2</li>
    <li>lista 3</li>
  </ul>
</div>
<script>
  var $ul = document.querySelector("ul");
  var $lis = document.querySelectorAll("li");
  var i;

  // // $lis é NodeList, e não elemento. Uma maneira de
  // Adicionar evento a cada item da lista:
  // for (i=0; i < $lis.length; i++){
  //   $lis[i].addEventListener("click",function(e){
  //     e.stopPropagation();
  //     console.log("cliqueu na li");
  //   });
  // }

</script>
```

Uma outra maneira é incluir os eventos na <ul> e delega-los às <li>. Da seguinte forma:

```
$ul.addEventListener("click", function(e){
  e.stopPropagation();
  console.log(e.target); //informa o elemento interno
                          //que foi clicado
  console.log(e.target.nodeName); //tipo HTML

  if (e.target.nodeName === "LI"){
    console.log("cliqueu em uma <li>");
  }
});
```

## 18.7 VALIDAÇÃO DE FORMULÁRIO, COM O BOTÃO SUBMIT:

- criar uma instância do <form>
- Adicionar função de validação pelo evento **submit**, através de addEventListener.
- Para interromper o envio, precisa interromper o evento (através do parâmetro da função, com o método **preventDefault()**).

Exemplo:

```
<script>
    // var $btn = document.querySelector("#btnEnviar");
    var $form = document.querySelector("form");

    $form.addEventListener("submit", ValidarCampos)
    // $btn.addEventListener("click", ValidarCampos);

    function ValidarCampos(e){
        $login = document.getElementById("txt_login");
        $senha = document.getElementById("txt_senha");
        if (($login.value === "") || ($senha.value === "")){
            alert("Login ou senha não podem ser vazios!");
            e.preventDefault();
        }else{
            console.log($login.value);
            console.log($senha.value);
        }
    }
</script>
```

## 18.8 ATRIBUINDO EVENTOS A NODELIST:

Não é possível adicionar eventos para Nodelist. Portanto, a saída é desenvolver um loop e adicionar o evento a cada elemento.

Exemplo:

```
<script>
    $btn = document.querySelectorAll(".botao-node");
    i=0;

    console.log($btn);
    console.log($btn.length);

    for (i=0; i < $btn.length;i++){

        $btn[i].addEventListener("click", function(){console.log(
this.textContent)})
    }
</script>
```

## 19. Strings em Javascript:

### 19.1 Declaração:

- **Literais:** var meuTexto = "Olá, mundo!";
- **Objetos:** var meuTexto = new String("Olá mundo!");

### 19.2 Métodos e Propriedades:

- **length:** tamanho da string (quantidade de caracteres). Sintaxe: <variável>.length;
- **replace():** substituição de caracteres. Substitui apenas o PRIMEIRO caractere. Para substituir todas, será necessário passar uma expressão regular.  
Sintaxe: <variável> = <string>.replace("<caracter\_a\_substituir>", "<novo\_caracter>");
- **indexOf():** retorna a posição de um determinado caractere, no sentido do início para o fim.  
Sintaxe: <string>.indexOf("<string\_buscada>");
- **lastIndexOf():** retorna a posição de um determinado caractere, no sentido do fim para o início.  
Sintaxe: <string>.lastIndexOf("<string\_buscada>");
- **slice():** Retorna parte da string, contando a partir da posição indicada (incluída no retorno) até a posição final (excluída do retorno). Não informar a posição final irá retornar da posição inicial até o final da string.  
Sintaxe: <string>.slice(<posição\_inicial>, [<posição\_final>]);
- **substring():** Semelhante a slice(). A diferença é quando se passa números invertidos. Se a posição final for informada como primeiro parâmetro, o substring faz a inversão automaticamente. O método slice retornaria uma string vazia neste caso.
- **split():** Divide a string de acordo com o delimitador, retornando um array.  
Sintaxe: <string>.split("delimitador");
- **toLowerCase():** Converte a string para caracteres minúsculos.  
Sintaxe: <string>.toLowerCase;

- **toUpperCase():** Converte a string para caracteres maiúsculos.

Sintaxe: <string>.toUpperCase;

- **valueOf():** retorna o valor primitivo da string.

Sintaxe: <string>.valueOf;

## 20. Números em Javascript (Objeto Number):

### 20.1 Declaração:

-**Literais:** var meuNum = 123;

-**Objetos:** var meuNum = new Number("123");

### 20.2 Métodos e Propriedades:

- **MAX\_VALUE:** Representa o valor máximo permitido para uma variável. Propriedade estática (propriedade do construtor, e não da instância). Portanto, aplicada apenas à **objetos** number, e não representam os valores das instâncias declaradas de forma literal.

Sintaxe: <variável>.MAX\_VALUE;

- **MIN\_VALUE:** Representa o valor mínimo permitido para uma variável. Propriedade estática (propriedade do construtor, e não da instância). Portanto, aplicada apenas à **objetos** number, e não representam os valores das instâncias declaradas de forma literal.

Sintaxe: <variável>.MIN\_VALUE;

- **toFixed:** Arredondamento para uma determinada quantidade de casas decimais. Se a quantidade de casas decimais for maior do que a já existente, será completado com 0.

Sintaxe: <variável>.toFixed(<qtd\_casas\_decimais>);

- **toPrecision():** Retorna apenas a quantidade de dígitos informada como parâmetro.

Sintaxe: <variável>.toPrecision(<qtd\_de\_digitos>);

Exemplo:

n = 12.345;

n = n.toPrecision(3);

//Retorna: n= 12.3;

- **toString():** Transforma um número em string. Pode-se utilizar para transformar os números de base decimais para binários ou hexadecimais, como por exemplo passando 2 ou 16 como parâmetros.

Sintaxe: <variável>.toString();

- **toExponential()**: Retorna a notação científica de um número. O parâmetro (opcional) indica quantas casas decimais após a parte inteira será apresentada na notação exponencial.

Sintaxe: <variável>.toExponential();

## 21. Objetos MATH:

- **min()**: Retorna o menor número entre os elementos passados por parâmetro.

Sintaxe: **Math.min**(<valor1>, <valor2>, <valor3>,...);

- **max()**: Retorna o maior número entre os elementos passados por parâmetro.

Sintaxe: **Math.max**(<valor1>, <valor2>, <valor3>,...);

- **round()**: Retorna o inteiro, com arredondamento para o mais próximo.

Sintaxe: **Math.round**(<valor>);

- **floor()**: Retorna o inteiro, com arredondamento sempre para baixo.

Sintaxe: **Math.floor**(<valor>);

- **ceil()**: Retorna o inteiro, com arredondamento sempre para cima.

Sintaxe: **Math.ceil**(<valor>);

- **pow()**: Retorna a potência. O primeiro argumento é a base, o segundo argumento é potência.

A mesma coisa de **\*\*** (operador aritmético).

Sintaxe: **Math.pow**(<base>, <potência>);

- **sqrt()**: Retorna a raiz quadrada.

Sintaxe: **Math.sqrt**(<valor>);

- **PI()**: Número PI.

Sintaxe: **Math.PI**();

- **cbrt()**: Retorna a raiz cúbica.

Sintaxe: **Math.cbrt**(<valor>);

- **random()**: Retorna um número aleatório entre 0 e 0.999999999 (quase 1).

Sintaxe: **Math.random**();

## 22. Arrays em Javascript:

### 22.1 Propriedades e métodos:

- **length**: quantidade de índices que possui um array.

Sintaxe: <array>.length;

- **every()**: Retorna um booleano, com o resultado de certa verificação. Recebe por parâmetro uma **função** que irá executar a verificação de todos os elementos. A função verificadora receberá 3 parâmetros: elemento, índice e objeto. O **return** deverá resultar SEMPRE em **true** ou **false**. Será **true** se TODOS os elementos passarem na verificação.

**Elemento**: obrigatório. É o elemento na posição do array (elemento no índice 0, elemento no índice 2, elemento no índice 3 e etc.).

**Índice**: Opcional. Índice de cada elemento (0, 1, 2, 3, etc.).

**Objeto**: Opcional. O conjunto (o próprio objeto array).

Sintaxe: <array>.every(function(<elemento>,[<índice>],[<objeto>]) {

Condições;

return true; //ou false

});

Exemplo:

```
var arr = [1,2,3,4,5];
var sohNumeros = arr.every(function(el){
    return typeof el === "number";
})
console.log(sohNumeros);
```

- **some()**: Semelhante ao método **every()**, porém retorna **true** se PELO MENOS UM elemento passar na verificação.

Sintaxe: <array>.some(function(<elemento>,[<índice>],[<objeto>]) {

Condições;

return true; //ou false

});

- **filter()**: Retorna um novo array, porém apenas com os elementos que satisfazem a alguma condição, definida pela função verificadora.

Sintaxe: <array>.some(function(<elemento>,[<índice>],[<objeto>]) {

Condições;

});

Exemplo:

```
var arr = ["sdc", 1,2,3,4,5];
```

```
var numeros = arr.filter(function(el, i, arr){
    return typeof el === "number";
})
```

- **forEach()**: Um loop, tal como o **for**, porém não é necessário incluir um contador. Também contém uma função, que será executada para cada elemento.

Sintaxe: <array>.forEach(function(el, i, obj){

...

});

- **indexOf()**: Retorna o índice de um determinado elemento. Se houver mais de um elemento com o valor procurado, irá retornar a posição do primeiro. Pode-se passar a posição inicial da busca como opcional. Caso o elemento não seja encontrado, irá retornar -1.

Sintaxe: <array>.indexOf("<elemento a ser localizado>", [posição\_inicial\_da\_busca]);

- **lastIndexOf()**: Semelhante ao `indexOf()`, porém começa a contar da direita para esquerda (irá mostrar a posição do último elemento encontrado, se houver mais de um com o mesmo valor procurado).

- **map()**: Acessa e altera cada um dos elementos de um array, conforme a função definida como parâmetro. A função agente também deverá ter os parâmetros elemento, e, optativamente, o índice e objeto.

Sintaxe: <array>.map(function(el, i, obj){

...

});

Exemplo:

```
var novoArrayNum = numeros.map(function(el, i, obj){
    return el * i;
    //retorna uma array
    //multiplica o elemento pelo seu índice
})
```

- **concat()**: Concatenar arrays. Pode-se passar como parâmetro elementos isolados para serem concatenados, ou um outro array previamente existente.

Sintaxe: <array>.concat(<array\_a\_ser\_concatenado>);



**- join():** Transforma um array em string (mais ou menos o oposto do split do objeto string).  
Optativamente, pode-se utilizar um parâmetro a ser utilizado como delimitador, em vez da vírgula padrão.

Sintaxe: <array>.join([delimitador]);

**- pop():** Remove o último elemento e devolve para quem chamou. Método destrutivo.

Sintaxe: var ultimoltem = <array>.pop();

**- push():** Acrescenta elementos no final de um array. Método destrutivo: =altera o array original, retornando o length da nova array.

Sintaxe: <array>.push("novo item1", "novo item2", "novo item3", etc.);

**- reduce():** Retorna um único valor, a partir de um array. Por exemplo: obter a soma dos elementos de um array. Recebe função, que pode conter parâmetros.

Sintaxe: var somaArr = <array>.reduce(function(<parametro1>, <parametro2>, ...){  
return <parametro1> + <parametro2> + ...;  
}, [valor\_inicial\_de\_parametro1] );

**- reduceRight():** Mesma operação do reduce(), porém inicia a operação da direita para a esquerda (do último elemento para o primeiro).

**- reverse():** Inverte a ordem dos elementos de um array. Método destrutivo.

Sintaxe: <array>.reverse();

**- shift():** Remove o primeiro elemento e devolve para quem chamou. Método destrutivo.

Semelhante ao pop().

Sintaxe: var primeiroItem = <array>.shift();

**- slice():** Retorna o pedaço de um array. Deve-se passar por parâmetro a posição inicial a partir de onde deve ser efetuado o corte (não inclui o elemento na posição informada), e onde termina (inclui o elemento da posição fim).

Sintaxe: array.slice(<posição\_inicial\_para\_corte>, [posição\_final\_para\_corte]);

**- splice():** Incluir, retirar elementos, em qualquer posição do array. Recebe por parâmetros:

1. Posição a partir de onde vai começar a transformação no array.
2. Quantidade de elementos que se deseja **remover** da array.
3. A partir do terceiro parâmetro, em quantidade ilimitada, os elementos que se deseja **incluir** na array.

Atenção: método destrutivo, que retorna os **índices removidos**. Se não remover nenhum elemento, retorna uma string vazia.

Sintaxe: <array>.splice(<posição\_inicial\_da\_alteração>, <qtd\_elementos\_exclusão>, <item1\_a\_incluir>, <item2\_a\_incluir>, <item3\_a\_incluir>, ...);

**- unshift():** Acrescenta elementos no início de um array. Método destrutivo: altera o array original, retornando o length da nova array.

Sintaxe: <array>.unshift("novo item1", "novo item2", "novo item3", etc.);

**- toString():** Transforma um array em string (mais ou menos o oposto do split do objeto string).

Semelhante ao join(), porém não admite alterar o delimitador, que seguirá como vírgula.

Sintaxe: <array>.toString();

**22.2 Spread Operator:** Serve para transformar elementos de um array em argumentos quando executada uma determinada função. Representado por reticências. Não dá suporte em browsers mais antigos.

Sintaxe: <função>(...<array>); //cada elemento do array é passado como parâmetro.

## 23. Objeto Date:

Sempre necessário ser instanciado. Portanto, a criação do objeto requer a instrução **new**.

Sintaxe: var <instância> = new Date([ano], [mês], [dia], [hora], [minutos], [segundos], [milissegundos]);

Exemplo:

Var minhaData = new Date();

**- getDay():** Retorna o dia da semana (de 0 a 6, representando de domingo a sábado).

**- getDate():** Retorna o dia do mês.

**- getFullYear():** Retorna o ano corrente.

**- getHours():** Retorna a hora atual no momento em que o objeto foi instanciado.

- **getMilliseconds()**: Retorna os milissegundos no momento em que o objeto foi instanciado.
  - **getMinutes()**: Retorna o minuto no momento em que o objeto foi instanciado.
  - **getMonth()**: Retorna o mês atual, de 0 a 11 (de janeiro a dezembro).
  - **getSeconds()**: Retorna os segundos no momento em que o objeto foi instanciado.
  - **getTime()**: Retorna o time stamping: a quantidade de milissegundos passados a partir de 01 de janeiro de 1970.
  - **getYear()**: retorna a quantidade de anos passados desde 1900.
- Pode-se atribuir valores para estas propriedades, utilizando o método **set** em vez de **get**.

## 24. Intervalos (Temporizadores):

Não são nativos do ECMAScript, de forma que são suportados pelo objeto **window** do browser (que não precisa ser referenciada no código). Não estão disponíveis no nodejs.

Como o Javascript é single thread, as execuções simultâneas serão armazenadas em uma fila de processos. Execuções simultâneas podem ocorrer pela definição de mais de um intervalo e também pela ocorrência de eventos.

- **setTimeout(fn, ms)**: executa uma função uma única vez após um determinado tempo.

Sintaxe: `setTimeout(<função>,<tempo_em_milissegundos>);`

- **clearTimeout(id)**: limpar uma função que foi agendada pelo `setTimeout`..

- **setInterval(fn, ms)**: cria um intervalo entre a execução da função, que se repetirá até que seja interrompida. Para interromper a execução, é preciso que a função `setInterval()` tenha sido atribuída a uma variável, que receberá o **id** do intervalo.

- **clearInterval(id)**: limpar uma função que foi agendada pelo `setInterval()`.

## 25. BOM(Browser Object Model):

- Propriedades e métodos relacionados ao browser.

### 25.1 Objeto Window:

Não está descrito nas ECMAScript.

- **innerWidth / innerHeight**: tamanho interno do browser (área disponível da página).

- **outerWidth / outerHeight**: tamanho externo do browser (área disponível da página).

- **opener**: retorna a página que chamou outra página através do método **open**.

- **pageXOffset / pageYOffset**: O quanto de scroll que a página está rolada.

- **screenX / screenY**: Posição da janela em relação ao monitor.

### 25.2 Objeto History: relacionado ao histórico de navegações.

**.length**: quantidade de páginas navegadas.

**.go(<índice>)**: navega na lista de históricos. Valores negativos retrocede, e positivos avançam.

**.back()**: volta uma página.

**.forward()**: avança uma página.

### 25.3 Objeto Location: informações da URL da página que está sendo visualizada.

#### location

**.hash**: Retorna tudo que estiver após o sinal # na URL.

**.host**: Retorna o servidor e a porta.

**.hostname**: Retorna o nome do servidor.

**.href**: retorna a URL em que se está navegando.

**.pathname**: retorna o endereço da página (excluído o servidor).

**.port**: retorna a porta.

**.protocol**: retorna o protocolo.

**.search**: retorna o que tiver após ? (informações passadas na URL).

**.assign(<url>):** Método. Navega para a <url>.

**.reload:** Método. Recarrega a página (refresh).

**.replace(<url>):** Método. Navega para <url>. Substitui a página atual. Não altera o histórico.

**25.4 Objetivo Navigator:** retorna informações sobre o browser.

**navigator**

**.appName:, .appVersion:, .userAgent:, .platform:, .cookieEnabled:, .language:, .mimeTypes:,**

**.plugins:**

**25.5 Objeto Screen:** referente ao monitor (tela).

**.availableWidth:**

**.availableHeight:**

**.width:**

**.height:**

**25.6 BOM: Métodos:**

- **alert("<mensagem>"):** Emite <mensagem> de alerta.
- **prompt("<mensagem>"):** Emite <mensagem> solicitando informação.
- **confirm("<mensagem>"):** Emite <mensagem> solicitando confirmação.
- **open("<URL>", ["id"], ["propriedades"]):** Abre uma janela (aba) com a <URL>.
- **close():** fecha a janela, exclusivamente se tiver sido aberta via método open.
- **scrollTo(x,y):** rolar a página até uma determinada posição fixa.
- **scrollBy(dx, dy):** rolar a página tantas vezes forem definidas nos parâmetros.
- **print():** abre a janela para impressão.

**25.7 BOM: Eventos**

- **Load:** quando todos os elementos da página são carregados.

**DOMContentLoaded:** ocorre quando a estrutura do HTML tiver completa, ainda que nem todos os elementos (imagens, por exemplo) tenham sido carregadas.

- **Error:** quando ocorre erro. Geralmente utilizado quando se vai carregar em iFrame.

- **Resize:** quando a janela tem a dimensão alterada.
- **Scroll:** ocorre quando sobe ou desce a barra de rolagem.
- **Unload:** ocorre quando a página é fechada. Não funciona alerts.
- **Beforeunload:** ocorre momento antes da página ser fechada. Não funciona alerts.

## 25.8 Estilizações (getComputedStyle()):

### Acessando estilos do CSS:

A propriedade **style** no Javascript apenas consegue acessar ou atribuir os estilos **inline** dos elementos. Caso um estilo tenha sido definido através de classes, o JS não consegue enxergar. Para acessar as propriedades **style** definidas como estilos no CSS, utiliza-se o **getComputedStyle()**.

**Sintaxe:** window.getComputedStyle(<elemento>).<propriedade>;

Browsers diferentes podem dar resultados diferentes.

## 25.9 Acessando Posições e Tamanhos em Relação ao Document:

Propriedades offset:

- offsetHeight, offsetWidth, offsetLeft, offsetTop.

Sintaxe:

<instância>.<propriedade\_offset>;

## 25.10 Acessando Posições e Tamanhos em Relação ao Viewport:

Sintaxe: <instância>.getBoundingClientRect();

Retorna uma NodeList com as informações relativas ao ViewPort.

## 26. Controles de Fluxo: Return, Break e Continue:

- **return:** sai da função, atribuindo valor, se definido.
- **break:** sai de um loop.
- **continue:** pula uma execução do loop, porem continua.

## 27. Conceito Progressive Enhancement:

Metodologia onde se inclui todo o HTML (e deixa funcionando), depois o CSS e por fim o JavaScript. Estrutura que permita o funcionamento mesmo com o HTML apenas.

## 28. Projeto Final – Anotações

### 28.1 Ferramentas de Desenvolvimento e Bibliotecas de Suporte:

- Chrome Web Developer (adicionar extensões do Chrome);
- <https://fontawesome.com/>
- <https://fonts.google.com/>
- Normalize.css

### 28.2 Técnicas:

**- Para saber se o JavaScript está ativo:** inclui uma classe “no-js” no <body>, e substitui, via JavaScript, por “js”. Se o JS estiver funcionando, a classe no <body> será substituída.

- Desenhar inicialmente para o formato mobile. Depois, utilizando Medias Queries, fazer as adaptações para os demais formatos de tela.

- Para organizar seções, como os cards, é melhor utilizar o Grid. Para organizar elementos, é melhor utilizar o Flexbox.

**- Desenhar seta na parte inferior da capa:** incluir 3 backgrounds. Os dois primeiros como um gradiente partindo da cor (branco) para transparente, com lados iguais (40px), sem repetição, com inclinação na diagonal e posição embaixo e central. O terceiro, é o background realmente de fundo.

```
background: linear-
gradient(45deg, white 50%, transparent 50%) left calc(50% + 20px) bottom
no-repeat,
linear-gradient(-
45deg, white 50%, transparent 50%) left calc(50% - 20px) bottom no-
repeat,
url(../img/01.jpg);
background-size: 40px 40px, 40px 40px, cover;
```

## **29. Servidores Para Publicação e Registro de Domínio:**

### **Hospedagens gratuitas:**

- [www.netlify.com](http://www.netlify.com)
- <https://www.000webhost.com/>

### **Registro de domínio:**

- [registro.com](http://registro.com) : para registros .com.br.
- [godaddy.com](http://godaddy.com): para registros .com.

**Se necessário, lembrar de configurar DNS e subdomínio, para reconhecer o www.**