

# Layouts com CSS

Disciplina: Desenvolvimento Web

Prof. Dr. Rafael Will M. de Araujo



# Sumário

- Fazer layouts com o CSS
- Dimensões no CSS
- Como funciona o *display*
- Como funciona uma *media query*
- Dois métodos de layout modernos: *Flexbox* e *Gridlayout*

## Conteúdo

# Dimensões

- Estas propriedades definem o tamanho dos elementos renderizados. Elas são height e width;
- Recebe valores em tamanho e porcentagem (mesmas unidades vistas anteriormente).

## Exemplo

```
1 seletor {  
2   height: valor;  
3   width: valor;  
4 }
```

# Dimensões

- Exemplo:

## CSS

```

1  div {
2    border: 1px solid black;
3  }

4  div.div1 {
5    height: 200px;
6    width: 25%;
7    background: yellow;
8  }

9  div.div2 {
10   height: 100px;
11   width: 5em;
12   background: red;
13 }
    
```

## HTML

```

1  <div class="div1">div 1</div>
2  <div class="div2">div 2</div>
    
```

**Exemplo:** <https://codepen.io/rafaelwill/pen/JjJEeXO>

# Classificação

- Estas propriedades são usadas para classificar os elementos e definir como eles devem aparecer no layout;
  - ▷ *float*
  - ▷ *clear*
  - ▷ *display*

# Float

- Define onde um elemento aparecerá em outro elemento;
- Pode assumir os valores *left* ou *right*;
- O elemento com float irá mover-se mais a esquerda ou mais a direita possível.

## Exemplo

```
1 seletor {  
2     float: valor;  
3 }
```

## CSS

## HTML

**Exemplo:** <https://codepen.io/rafaelwill/pen/oNwBQeM>



# Clear

- Define em quais lados de um elemento outros elementos não podem flutuar (*float*);
- Assume valores: *none*, *left*, *right*, *both*.

## Exemplo

```
1 seletor {  
2     clear: valor;  
3 }
```

- Exemplo:

CSS

```

1  .div1 {
2      float: left;
3      background-color: gray;
4  }

5  .div2 {
6      float: left;
7      clear: left;
8      background-color: cyan;
9  }
```

HTML

```

1  <div class="div1">div 1</div>
2  <div class="div2">div 2</div>
```

**Exemplo:** <https://codepen.io/rafaelwill/pen/QWgdJqy>

# Display

- Define a maneira como os diversos elementos de uma página são exibidos. Ele especifica o comportamento que o elemento terá no momento que for "renderizado".

## Exemplo

```
1 seletor{
2     display: valor;
3 }
```

- Temos quatro valores básicos:
  - ▷ **block**: define uma região em bloco, que possui altura definida pelo conteúdo (ou CSS) e se expande da esquerda para a direita até onde for possível.
    - ◊ O elemento começará em uma nova linha e ocupará toda a largura disponível. Permite definir os valores de largura e altura.
  - ▷ **inline**: define uma região em linha, com a largura definida pelo conteúdo e altura pelo tamanho da linha.
    - ◊ O elemento não começa em uma nova linha e ocupa apenas a largura necessária. **Não é possível definir a largura ou altura.**
  - ▷ **inline-block**: define uma região com tamanho controlada pelo conteúdo ou CSS. Nenhuma *tag* funciona assim por padrão.
    - ◊ É formatado como um elemento inline, isto é, ele **não começa em uma nova linha**. Mas, **permite definir valores de largura e altura**.
  - ▷ **none**: define uma região invisível, ela existe no HTML mas não aparece para o usuário e não altera o layout.

# Display

- Exemplo: <https://codepen.io/rafaelwill/pen/WNORYXQ>

## CSS

```

1  .invisible {
2      display: none;
3  }

4  .block {
5      display: block;
6      background-color: gray;
7  }

8  .inline {
9      display: inline;
10     background-color: cyan;
11 }
```

## HTML

```

1  <div class="invisible">div invisible</div>
2  <div class="block">div block</div>
3  <div class="inline">div inline</div>
4  <div class="inline">outra div inline</div>
```

# Display

- Exemplo 2: <https://codepen.io/rafaelwill/pen/dyRNQZg>

## CSS

```

1  div {
2    border: 3px solid black;
3    width: 100px;
4    height: 100px;
5  }

6  div.div1 {
7    background: yellow;
8    display: block;
9  }

10 div.div2 {
11   background: red;
12   display: inline;
13 }

14 div.div3 {
15   background: green;
16   display: inline-block;
17 }

18 div.div4 {
19   background: black;
20   display: none;
21 }
    
```

## HTML

```

1  <div class="div1">div 1</div>
2  <div class="div2">div 2</div>
3  <div class="div3">div 3</div>
4  <div class="div4">div 4</div>
    
```

## Conteúdo

- 1 Introdução
- 2 **Box-Model**
- 3 Media Query
- 4 Layouts
- 5 Exercício

# Box-Model

- Além da altura (*height*) e largura (*width*), temos algumas outras medidas que influenciam no tamanho total que um elemento HTML ocupa na página:
  - ▷ **Border**: já mostramos a borda, ela possui uma espessura, o que altera também o espaço ocupado pela *tag* HTML;
  - ▷ **Padding**: espaço entre o conteúdo da *tag* e a borda dela. Pode ser colocado para as quatro direções ou apenas as que forem interessantes (*top*, *bottom*, *left*, *right*);
  - ▷ **Margin**: espaço entre a borda do elemento até a borda do próximo elemento (ou de algum limite do documento). Também pode ser utilizada em todas as quatro direções ou apenas nas que forem interessantes.
- Exemplo:

# Box-Model: border

- Exemplo: <https://codepen.io/rafaelwill/pen/jOwyXrL>

## CSS

```
1  div {  
2    border: 3px solid black;  
3    width: 100px;  
4    height: 100px;  
5  }  
  
6  div.div1 {  
7    background: yellow;  
8  }  
  
9  div.div2 {  
10   background: red;  
11   border-width: 10px;  
12 }
```

## HTML

```
1  <div class="div1">div 1</div>  
2  <div class="div2">div 2</div>
```



# Box-Model: padding

- Exemplo: <https://codepen.io/rafaelwill/pen/oNwBJLr>

## CSS

```
1  div {
2    border: 3px solid black;
3    width: 100px;
4    height: 100px;
5  }

6  div.div1 {
7    background: yellow;
8    padding: 10px;
9  }

10 div.div2 {
11   background: red;
12   padding: 20px;
13 }
```

## HTML

```
1  <div class="div1">div 1</div>
2  <div class="div2">div 2</div>
```

# Box-Model: margin

- Exemplo: <https://codepen.io/rafaelwill/pen/abwpPBP>

## CSS

```

1  div {
2    border: 3px solid black;
3    width: 100px;
4    height: 100px;
5  }

6  div.div1 {
7    background: yellow;
8    margin: 10px
9  }

10 div.div2 {
11   background: red;
12   margin: 20px;
13 }
```

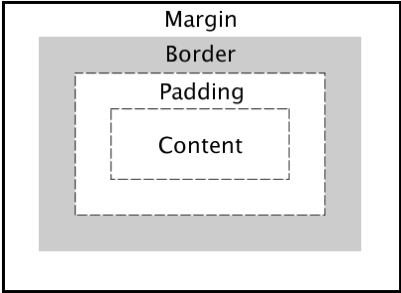
## HTML

```

1  <div class="div1">div 1</div>
2  <div class="div2">div 2</div>
```

# Box-Model

- Essas distâncias configuram o que chamamos de Box-Model (modelo caixa):



- Desta maneira, quando definimos os atributos *height* e *width* do elemento em si, esses valores não levam em consideração o *padding* e o *border*, fazendo com que os elementos sejam maiores do que deveriam.

# Box-Model

- Exemplo: <https://codepen.io/rafaelwill/pen/yLXgGMK>

## CSS

```
1 .div1 {  
2   width: 300px;  
3   height: 100px;  
4   border: 1px solid blue;  
5 }  
  
6 .div2 {  
7   width: 300px;  
8   height: 100px;  
9   padding: 50px;  
10  border: 1px solid blue;  
11 }
```

## HTML

```
1 <div class="div1">div menor (width: 300px e height: 100px)</div>  
2 <div class="div2">div maior (width: 300px e height: 100px)</div>
```

- Note que a div2 será maior que a div1. Isso ocorre porque, por padrão, o CSS não considera *padding*, *border* e *margin* no cálculo final do tamanho do elemento.

- CSS

## HTML

**Exemplo:** <https://codepen.io/rafaelwill/pen/vYZgvmO>

- Uma atitude comum entre os desenvolvedores é utilizar o *border-box* para todos os objetos como padrão. Para facilitar, podemos utilizar a seguinte regra para **colocar esse atributo para todos os elementos da página**:

## Exemplo

```
1 * {  
2   box-sizing: border-box;  
3 }
```



# Media Query

- O conceito principal de uma *Media Query* é verificar algumas informações do dispositivo visualizando o site, assim podendo definir um conjunto de regras que funcione melhor para este dispositivo;
- É possível verificar principalmente:
  - ▷ Largura e altura do *viewport* (área visível da página);
  - ▷ Largura e altura do aparelho em si (tela completa);
  - ▷ Orientação (retrato ou paisagem);
  - ▷ Resolução
- Técnica de *Media Query* é muito popular no desenvolvimento mobile;
- Sintaxe básica:

## Exemplo

```
1 @media not|only tipomedia and (mediafeature) {  
2     Regras CSS  
3 }
```

- Mais informações: [https://developer.mozilla.org/pt-BR/docs/Web/CSS/Media\\_Queries/Using\\_media\\_queries](https://developer.mozilla.org/pt-BR/docs/Web/CSS/Media_Queries/Using_media_queries)



# Media Query

- Neste exemplo, as áreas das classes `div1` e `div2` possuem um tamanho especificado.
- Quando o aparelho possuir resolução de tela (*screen*) com largura máxima de 700px, os tamanhos passam a ser definidos por 100% (tamanho total da janela).
- O marcador *only* diz que essa regra se aplica **apenas** ao *mediatype* (*screen*) e *mediafeature* (*max-width: 700px*).

## CSS

```

1  .div1 {
2    width: 200px;
3    background: cyan;
4  }

5  .div2 {
6    width: 1000px;
7    background: yellow;
8  }

9  @media only screen and (max-width: 700px) {
10   .div1 {
11     width: 100%;
12   }

13   .div2 {
14     width: 100%;
15   }
16 }
```

## HTML

```

1  <div class="div1">div 1</div>
2  <div class="div2">div 2</div>
```



# Media Query

- continuação do Exemplo 2:

## HTML

```
1 <div>
2   <div id="leftsidebar">
3     <ul id="menulist">
4       <li class="menuitem">Item 1</li>
5       <li class="menuitem">Item 2</li>
6       <li class="menuitem">Item 3</li>
7       <li class="menuitem">Item 4</li>
8       <li class="menuitem">Item 5</li>
9     </ul>
10  </div>
11  <div id="main">
12    <h1>Redimensione o navegador e veja o que acontece!</h1>
13    <p>Este exemplo mostra um menu a esquerda do conteúdo principal quando
14    o site tem pelo menos 480px de largura. Experimente ver o que acontece quando a tela fica menor.</p>
15  </div>
16 </div>
```



# Flexbox

- No CSS3 foi definida mais uma propriedade *display*, a **Flex(box)**. A ideia do Flexbox é garantir que o conteúdo interno da página consiga se reorganizar de uma maneira previsível caso a janela mude de tamanho;
- Ajuda a fazer layouts de maneira mais simplificada e fluída;
- Propriedades que fazem parte do modelo flexbox são divididas em propriedades para o container (que possui o display **flex**, chamado de **flex-container**) e para os seus filhos (chamados de **flex-items**);

- 

- 30 / 41

# Grid Layout

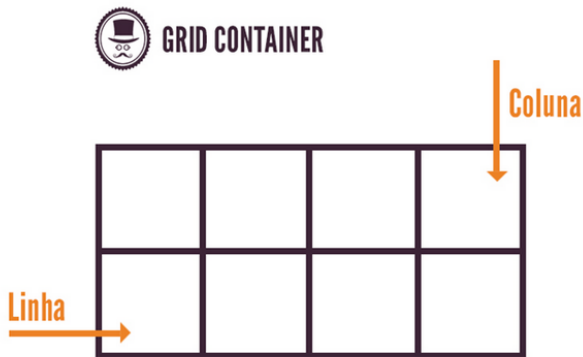
- Apesar do Flexbox já ter adicionado uma ótima possibilidade de layouts fluídos, ele funciona apenas para containers em uma dimensão (linha ou coluna);
- Para obter layouts mais complicados, é necessário combinar vários containers com flexbox;
- Seria interessante possuir um sistema que funcionasse como uma tabela do HTML (definição de regiões), mas que não dependesse de tabelas do HTML;
- A partir do CSS3 criou-se o Grid-Layout, uma forma de construir o layout como se fosse uma tabela, ou um grid, sem usar as tabelas ou outros "hacks" de CSS;
- Vale lembrar que a sua utilização é um tanto restrita: nem todos os navegadores implantaram toda sua especificação (leia-se: Opera e Edge).

# Grid Layout

- Assim como no Flexbox, vamos dividir as propriedades do GridLayout em propriedades do **container** (elemento pai do layout) e propriedades dos **grid-itens** (itens desse container);
- O GridLayout é bem extenso, portanto vamos cobrir apenas o básico das suas propriedades (você pode consultar mais em: <https://css-tricks.com/snippets/css/complete-guide-grid>)
- As definições da GridLayout são:
  - ▷ **Grid**: área que visualmente possui linhas e colunas (como se fosse tabela);
  - ▷ **Grid Line**: linhas divisórias que estruturam o grid;
  - ▷ **Grid Item**: item interno da grid (qualquer tag HTML);
  - ▷ **Grid Track**: espaço entre as linhas (vertical ou horizontal);
  - ▷ **Grid Area**: qualquer espaço entre quatro linhas dentro de um grid.



# Grid Layout

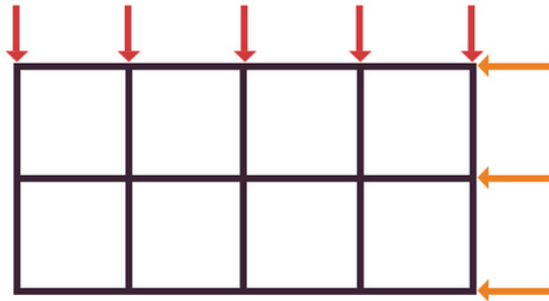


- Créditos das imagens: <http://www.chiefofdesign.com.br/css-grid-layout>

# Grid Layout



## GRID LINES



- Créditos das imagens: <http://www.chiefofdesign.com.br/css-grid-layout>

# Grid Layout



## GRID ITEM / GRID CELL

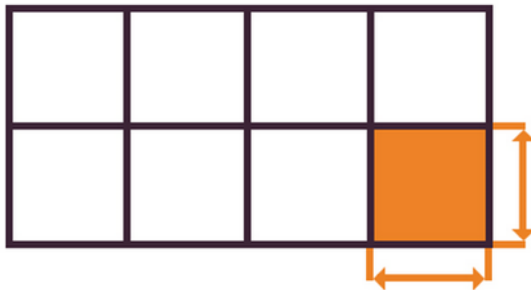


- Créditos das imagens: <http://www.chiefofdesign.com.br/css-grid-layout>

# Grid Layout



GRID TRACKS



- Créditos das imagens: <http://www.chiefofdesign.com.br/css-grid-layout>

# Grid Layout



- Créditos das imagens: <http://www.chiefdesign.com.br/css-grid-layout>

## Grid Layout

- Dentro das definições do Grid, temos as seguintes propriedades para o **container**:
  - ▷ **Grid-template-columns**: Define quantas colunas e o tamanho básico de cada uma;
  - ▷ **Grid-template-rows**: Define quantas linhas e o tamanho básico de cada;
  - ▷ **Grid-template-areas**: Define área nomeadas por classe do grid;
  - ▷ **Grid-row-gap**: Espaço entre linhas;
  - ▷ **Grid-column-gap**: Espaço entre colunas.

# Grid Layout

- Dentro das definições do Grid, temos as seguintes propriedades para os **itens**:
  - ▷ `Grid-column|row-start`: Determina em qual coluna ou linha o item começa;
  - ▷ `Grid-column|row-end`: Determina em qual coluna ou linha o item termina;
  - ▷ `Grid-area`: Determina qual área nomeada (classe) do grid esse item vai ocupar;
- Para entender mais sobre o grid, veja: <http://cssgridgarden.com>

## Conteúdo

- 1 Introdução
- 2 Box-Model
- 3 Media Query
- 4 Layouts
- 5 Exercício



# Exercício

- Baixe o arquivo com o logotipo da empresa fictícia "Cursinho Web" e outras imagens atualizadas com fundo transparente. Em seguida, faça:
  - ▷ Adicione a imagem do logotipo ao início do *header* com o *id* "img-logo" e remova as barras verticais entre os links dentro do nav. Isso deve ser replicado em todas as páginas HTML.
  - ▷ Adicione a propriedade *margin* com valor 0px no seletor do *body*.
  - ▷ Adicione as seguintes propriedades ao seletor *header*:
    - ◇ Altura (*height*) 80px;
    - ◇ Margem (*margin*) 0px;
    - ◇ Padding (*padding-right*) à direita de 1rem;
    - ◇ *display flex*;
    - ◇ *justify-content* com valor *space-between*;
    - ◇ *align-items* com valor *center*;
  - ▷ Adicione um seletor para a tag *main* com os seguintes valores:
    - ◇ Margem esquerda: 1rem;
    - ◇ Margem direita: 1rem;
    - ◇ Margem do topo: 0px;
    - ◇ Margem do fundo: 100px;
  - ▷ Adicione as seguintes propriedades ao seletor *footer*:
    - ◇ Alinhamento do texto: centralizado;
    - ◇ Comprimento (*width*) 100%;
    - ◇ Propriedade *position* com o valor *fixed* (força o *footer* a ficar num lugar fixo, ao invés de seguir a sequência de tags da página);
    - ◇ Propriedade *bottom* com valor 0 (força o *footer* a ficar preso na parte de baixo da página).
    - ◇ Padding de 5px;
  - ▷ Mude a margem do topo para 0px no seletor *h1*.

# Exercício

- (continuação):
  - ▷ Adicione as seguintes propriedades a todos os links dentro do *nav*:
    - ◇ Cor de fundo: #ededed;
    - ◇ Borda: 1px, borda sólida com cor #bbbbbb;
    - ◇ Padding de 5px;
    - ◇ Margem: 1px;

## Exercício: *index.html*

[illegible]

# Exercício: disciplinas.html

Cursinho Web - Disciplinas
+

File | /home/will/tmpfsDisk/site\_teste\_set/disciplinas.html

Cursinho Web

Página inicial
Disciplinas oferecidas
Entre em contato

Disciplinas oferecidas:

Desenvolvimento Web

Conhecer os conceitos de desenvolvimento de sistemas dinâmicos para a Internet, aplicando as mais recentes tecnologias e métodos para esta área.

Habilidades

- HTML
- CSS
- JavaScript
- Python e Flask

HTML

CSS

JS

Programação Orientada a Objetos

Desenvolver lógica de raciocínio para organizar os elementos necessários para o desenvolvimento de programas orientado a objetos.

- Dicionários
- Tratamento de exceções
- Testes unitários
- Construção de classes, atributos e métodos
- Herança e polimorfismo


python

Cursinho Web - Todos os direitos reservados  
Contato: email@provedor.com

# Exercício: contato.html

Cursinho Web - Contato

File | /home/will/tmpfsDisk/site\_teste\_set/contato.html

 Cursinho Web

Página inicial

Disciplinas oferecidas

Entre em contato

Dados para contato:

Nome completo:

E-mail:

Qual disciplina te interessa mais?

Desenvolvimento Web

Quais linguagens você conhece?

☐ C#

☐ Java

☐ JavaScript

☐ Python

☐ PHP

☐ Ruby

Onde você conheceu o Cursinho Web?

☐ Amigos

☐ Google

☐ Youtube

☐ Instagram

Limpar

Enviar

Cursinho Web - Todos os direitos reservados  
Contato: email@provedor.com