

**INSTITUTO
FEDERAL**

Paraíba

Campus
Campina Grande

4

Curso Técnico Subsequente em Informática
Desenvolvimento de Aplicações para Web

Cascading Style Sheets

Prof. César Rocha - Copyright©2023
cesarrocha@ifpb.edu.br



Roteiro

- Em poucos slides, explorar os fundamentos do padrão Web **CSS**
 - sintaxe, seletor, propriedades, regras, CSS no HTML, vantagens, herança e override de propriedades no código CSS, otimizando partes do CSS, declaração de classes, pseudo-classes, usando o validador CSS, etc.
- Mostrar vários exemplos de códigos CSS que você deve testar para solidificar seus conhecimentos nesta nova linguagem
 - Este novo assunto é, como no caso do HTML, de suma importância para as próximas disciplinas do curso (Desenvolvimento Web, etc.)
- Aconselha-se utilizar uma referência para acompanhar este assunto
 - Mais detalhes sobre qual referência usar são discutidos adiante...

O início da Web...

- Boa parte dos documentos publicados na Web eram concebidos utilizando regras oriundas do **HTML** (1.0, ..., 3.2, 4.01, ...)
 - HTML tinha **dupla função**: estruturar o conteúdo e aparência (estilizá-lo)
 - Há, ainda, uma grande variedade de tag's e atributos para estilização
 - Hoje, estas tag's e atributos constituem **código legado** (depreciado) pelo HTML
- Neste tipo de HTML, mudanças na página eram muito difíceis!
 - Não havia **reuso** de estilização para outras páginas
 - Não havia um consenso no tocante a que os navegadores **deviam de obedecer**
- Veja com seus próprios olhos...

O início da Web(cont.)

```
<html>
  <head>
    <title>Lojas Maia (ano 1998)</title>
  </head>
  <body bgcolor=tan text=black>
    <p align=center>Venha conhecer nossa <font face=arial>queima</font>
    de estoque da semana! Temos promoções imperdíveis para você! </p>
    <p> Veja alguns exemplos:
      <ul>
        <li> Liquidificador Arno - R$ 56.99
        <li> Aspirador de pó CCE - R$ 156.99
      </ul>
      <p align=right> Traga logo sua família! Não perca tempo.
      <center><font size="small"> Ofertas válidas enquanto
      durar o estoque!</font></center>
      <p align=center><font size="small">Copyright &copy; 1998 - todos os direitos são
      estritamente reservados.</font>
    </body>
</html>
```

Aqui estão alguns atributos que controlam a aparência. bgcolor define a cor de fundo e o text define a cor do texto existente

Tipo da fonte!

Alinhamento

Você podia escrever algumas tag's sem fechá-las (e torcia para que todos os browsers entendessem isso...)

Outra forma de alinhar!

Agora o tamanho da fonte!

Alguns atributos requeriam aspas duplas e outros não...

Obs.: também não havia nenhum DOCTYPE associado ao documento...

Um padrão de apresentação

CSS



- O padrão Web **CSS 3.0** (Cascading Style Sheets)
 - Na verdade, uma nova linguagem que possibilita a criação de **regras** de apresentação para uma página ou até um site completo
 - O CSS **precisa da definição da estrutura** do HTML para funcionar
 - Uma regra CSS é uma declaração que segue uma sintaxe própria e que define como será aplicado um estilo a um (ou mais) elemento(s) HTML
- Um conjunto de regras CSS forma uma **folha de estilos**
 - Estas regras são colocadas em um **arquivo texto** com extensão **.css**
 - Como o HTML, o CSS também **não depende** de uma ferramenta específica

Obs.: detalhes das propostas da W3C sobre o CSS (ainda não completamente suportadas por todos os navegadores atuais) podem ser vistas em: <http://www.w3.org/Style/CSS/current-work>



Sintaxe de uma regra

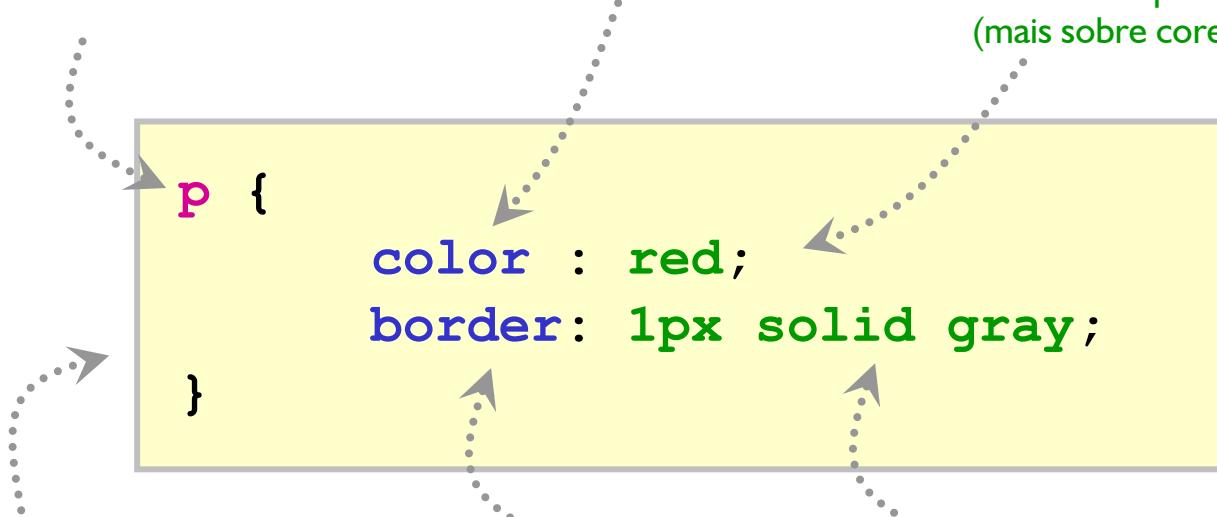
- Uma regra CSS, na sua forma mais elementar, é composta de três partes: um **seletor**, uma **propriedade** e um **valor**.

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

- Onde:
 - **Seletor**: genericamente, é o elemento HTML (ou pseudo-seletores, de classe e ID) para o qual a regra será aplicada;
 - **Propriedade**: é o “atributo” do elemento ao qual será aplicada a regra;
 - **Valor**: é a característica específica a ser assumida pela propriedade.

Exemplo:

A primeira coisa que se deve fazer é selecionar o elemento HTML que se quer estilizar. Note, porém, que não há sinais <> no seletor!



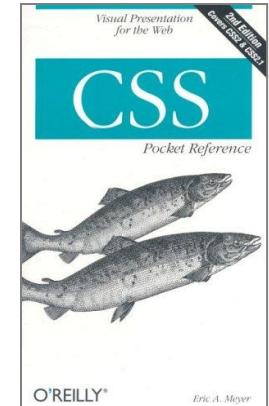
Todo este código é chamado de regra CSS. Ela deve existir, preferencialmente, dentro de um arquivo .css

Obs.: observe as posições das chaves, dos sinais de dois pontos e ponto e vírgulas!



Comentários

- A regra mostrada há pouco seria aplicada a todos os parágrafos
 - Ora, mas se fosse preciso aplicar duas cores diferentes a dois parágrafos?
 - Será visto que CSS é capaz de selecionar apenas um elemento (e até sub-elementos!)
 - E se houvesse um `` dentro do parágrafo? Sua cor seria alterada?
- Para utilizar CSS, faz-se necessária uma boa referência
 - Ora, como é possível saber quais são todas as propriedades, valores e sintaxe possíveis aplicáveis a um elemento HTML?
 - Ex.: é possível mudar a cor do texto de uma tag `` ?
- **CSS Pocket Reference** (Eric Meyer)



Ligando o CSS ao HTML

- Basicamente, há **duas formas** de se conectar um CSS a um HTML
 - ➊ Fazendo uso da tag `<style> ... </style>` na própria página

Exemplo:

```
<!doctype html>
<html lang="pt">
  <head>
    <meta charset="utf-8">
    <title>O título da página</title>
    <style>
      p {
        color : red;
        border: 1px solid gray;
      }
    </style>
  </head>
  ...

```

Para adicionar CSS diretamente em apenas um arquivo HTML, adicione a tag `<style>`

Escreva as regras CSS dentro de `<style>`

Obs.: esta não é, necessariamente, a melhor abordagem. Apenas um arquivo é estilizado e não há reuso de regras ou folhas de estilos completas para outras páginas que venham a surgir, futuramente.



Ligando o CSS ao HTML(cont.)

② Fazendo uma referência a um arquivo css externo à página

Exemplo:

```
<!doctype html>
<html lang="pt">
  <head>
    <meta charset="utf-8">
    <title>O título da página</title>

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

```
      <style>
        p{ ... }
      </style>
    </head> ...
```

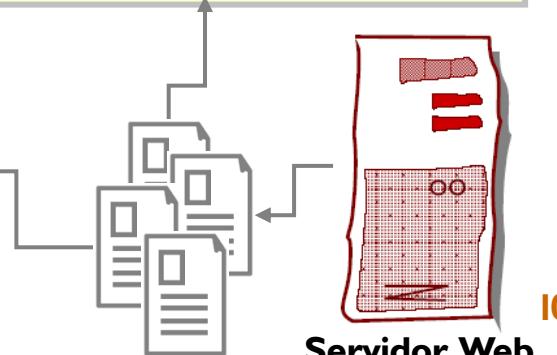
Delete isso...

Obs.: este arquivo css pode residir na sua estrutura de diretórios ou até mesmo vir de uma URL absoluta.

Aqui está o código HTML que indica ao browser onde encontrar o arquivo externo

`p {
 color: red;
 border: 1px solid gray;
}`

style.css



Anatomia de um <link>

O elemento <link>
informa ao browser que
uma informação externa
será “linkada” à página

A informação é “stylesheet”. Em outras palavras, o browser deve considerar o texto como sintaxe css.

A localização do arquivo é dita no atributo href (note que está sendo usada uma URL relativa, mas isso não é obrigatório!)

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

* Aqui vale a mesma regra para doctype: o <link> pode ser escrito em apenas uma linha. Contudo, se for necessário quebrar uma linha, esta deve ser feita entre atributos

O atributo rel identifica o relacionamento entre o arquivo HTML e a informação que está sendo linkada. Neste caso, estamos linkando uma folha de estilos

<link> é um elemento vazio. Não há uma tag final de fechamento ‘/’



Um exemplo breve...

Laboratório 07: definindo uma folha de estilos CSS



Otimizando o código CSS

- É possível definir quantas regras você quiser em um arquivo CSS
 - Inclusive, é possível utilizar uma mesma propriedade, várias vezes.
- Exemplo:

Aqui está uma regra para configurar a cor de todos os cabeçalhos de nível 1 `<h1>` na página, além de configurar o tipo da fonte para sans-serif

E aqui está uma regra para fazer a mesma coisa com todos os cabeçalhos de nível 2 `<h2>` encontrados

Mudar a cor de todos os parágrafos encontrados para vermelha

```
h1 {  
    color : gray;  
    font-family: sans-serif;  
}  
  
h2 {  
    color : gray;  
    font-family: sans-serif;  
}  
  
p {  
    color : red;  
}
```

The diagram illustrates how specific CSS rules can be applied to multiple elements. Three arrows originate from the explanatory text on the left and point to the corresponding CSS declarations in the code block on the right. The first arrow points to the `h1` rule, the second to the `h2` rule, and the third to the `p` rule.



Otimizando o código CSS (cont.)

- Porém, há um problema com a abordagem anterior...
 - Há **duplicação** de propriedades e valores!
- O que fazer?

Para escrever uma mesma regra para vários seletores, escreva-os separados por vírgula

Nesta regra, todos os cabeçalhos de níveis 1 e 2 terão suas aparências alteradas automaticamente...

Delete isso...

Pergunta intrigante...

...mas, o que faríamos se quiséssemos, amanhã, colocar uma linha de borda apenas nos cabeçalhos de nível 1 <h1>, mas não em cabeçalhos <h2>?

```
h1, h2 {  
    color : gray;  
    font-family: sans-serif;  
}  
  
h2 {  
    color : gray;  
    font-family: sans-serif;  
}+  
  
p {  
    color : red;  
}
```

arquivo.css



Regras aplicadas a um mesmo seletor

- Até aqui, cada regra mostrada mencionava sempre tag's distintas
 - Saiba que é permitido escrever várias regras **para uma mesma tag** HTML
- Assim, no geral, se houver regras que mencionam um mesmo descritor HTML, cada regra irá “adicionar” aparência em relação à anterior *
 - Pode-se “**acumular**” aparência escrevendo regras mais específicas

```
h1, h2 {  
    color : gray;  
    font-family: sans-serif;  
}  
  
h1 {  
    border-bottom : 1px solid black;  
}
```

* Veremos, mais adiante, que, em alguns casos, o browser irá decidir de acordo com o peso da regra (não considerando se ela é a última dentro do arquivo css).



Overriding (sobreposição)

- Em situações que envolvam regras aplicadas a um mesmo descritor HTML, é comum **sobrepor** a aparência definida em regras anteriores
 - A isso, dá-se o nome de *override* (ou sobreposição) de aparência
- Se as propriedades são conflitantes, será escolhida a regra mais específica que o browser encontrar dentro do arquivo CSS

```
h1, h2, h3 {  
    color : gray;  
    font-family: sans-serif;  
}  
  
h1 {  
    color : red;  
    border-bottom : 1px solid black;  
}
```

Obs: note que a cor cinza do cabeçalho h1 será sobreposta pela última regra mais específica



Um meta-padrão

- Procure **fatorar** código comum a todos os descritores HTML
 - Ora, se a aparência tiver de mudar, apenas um único local será alterado
- Só então, a partir daí, defina regras de estilo mais **específicas**
 - Uma boa dica é fazer *override* de propriedades, caso estas já tenham sido definidas em regras de estilo anteriores no código CSS
 - Mesmo assim, muito cuidado na escrita de regras conflitantes!
- E lembre-se: ao resolver um conflito de regras em um arquivo CSS, o browser irá sempre em busca da regra mais específica
 - Mais sobre o cálculo de pesos no final do módulo...

Herança

- Mesmo fatorando código comum e se valendo de override para ajustes específicos, em CSS, é facil começar a **duplicar propriedades**

Veja um exemplo:

Conforme visto em slides anteriores, é interessante fatorar código comum aos elementos (neste caso, os cabeçalhos `<h1>` e `<h2>`, suas cores e tipo de fonte)

Ajustes específicos em `<h2>` foram feitos com override, mas...

...problema! Ora, por que definir `font-family` para cada elemento? E se, amanhã, for adicionado à página um elemento `<blockquote>`? Teríamos de adicionar uma nova regra no arquivo css para ele também?

```
h1, h2 {  
    color : gray;  
    font-family: sans-serif;  
}  
  
h2 {  
    color : red;  
    border-bottom : 1px solid black;  
}  
  
p {  
    color : blue;  
    font-family: sans-serif;  
}
```

arquivo.css



Herança(cont.)

- “Elementos-filhos” podem herdar aparência de seus “pais”
 - Mas isso apenas para **algumas** propriedades*!

Iremos mover a propriedade para o “pai” de todos os elementos envolvidos no arquivo.

A grande maioria dos filhos de <body> (inclusive seus “netos”) serão afetados diretamente; como se houvesse uma regra explícita para cada um.

- Pergunta intrigante:

...mas, e se houvesse uma tag definida dentro de algum parágrafo? é um elemento inline (como <a>, ,...); sua fonte seria alterada?

```
body {  
    font-family: sans-serif;  
}  
  
h1, h2 {  
    color : gray;  
    font-family: sans-serif;  
}  
  
h2 {  
    color : red;  
    border-bottom : 1px solid black;  
}  
  
p {  
    color : blue;  
    font-family: sans-serif;  
}
```

Delete isso...

Delete isso...

arquivo.css



* sobretudo propriedades de texto, mas não de borda, por exemplo

“Sobrepondo” a Herança

- Movendo a propriedade para <body>, quase todos os elementos tiveram suas fontes afetadas

- Mas, e se houver uma herança indesejada?

Ora, você pode anular a herança fazendo override específico para um determinado elemento afetado por ela...

- Como faço isso?

É facil: apenas adicione uma nova regra selecionando apenas um determinado elemento desejado e atribuindo uma nova aparência ao mesmo

```
body {  
    font-family: sans-serif;  
}  
  
h1, h2 {  
    color : gray;  
}  
  
h2 {  
    color : red;  
    border-bottom : 1px solid black;  
}  
  
p {  
    color : blue;  
}  
  
em {  
    font-family: serif;  
}
```



“Sobrepondo” a Herança (cont.)

O Chrome Dev Tools ajuda a identificar herança e override de regras no CSS

The screenshot shows the Chrome Dev Tools interface with the 'Elements' tab selected. On the left, the DOM tree displays the following structure:

```
<!DOCTYPE html>
<html lang="en">
  <head> ...
  <body>
    <h1>Um cabeçalho nível 1</h1>
    <h2>Um cabeçalho nível 2</h2> == $0
    <p>"Um parágrafo com uma palavra "<br/><em>importante</em>
  </p>
  <!-- Code injected by live-server -->
  <script> ...
</body>
</html>
```

A tooltip labeled "Override!" points to the `color: red;` rule for `h2`. Another tooltip labeled "Indicação de herança" points to the `color: gray;` rule for `h1, h2`.

The 'Styles' tab in the Dev Tools is active, showing the following CSS rules:

```
element.style {
}
h2 {
  color: red;
  border-bottom: 1px solid black;
}
h1, h2 {
  color: gray;
}
h2 {
  display: block;
  font-size: 1.5em;
  margin-block-start: 0.83em;
  margin-block-end: 0.83em;
  margin-inline-start: 0px;
  margin-inline-end: 0px;
  font-weight: bold;
}
Inherited from body
body {
  font-family: sans-serif;
}
```

A large callout box highlights the `color: gray;` rule for `h1, h2`, with an arrow pointing to its declaration in the CSS file:

```
body {
  font-family: sans-serif;
}
h1, h2 {
  color: gray;
}
```

Another callout box highlights the `color : red;` and `border-bottom : 1px solid black;` rules for `h2`, with an arrow pointing to its declaration in the CSS file:

```
h2 {
  color : red;
  border-bottom : 1px solid black;
}
```

Callout boxes also point to the `color : blue;` rule for `p` and the `font-family : serif;` rule for `em`:

```
p {
  color : blue;
}
em {
  font-family: serif;
```

“Sobrepondo” a Herança (cont.)

O Chrome Dev Tools ajuda a identificar herança e override de regras no CSS

The screenshot shows the Chrome Dev Tools interface with the 'Elements' tab selected. On the left, the DOM tree displays an HTML document structure. A specific element, an *em* tag containing the text "importante", is highlighted. Dotted arrows point from this element to two annotations: "Indicação de herança" (inheritance indication) pointing to the 'Inherited from p' section in the styles panel, and "Override!" pointing to the 'body' rule in the styles panel.

The 'Styles' panel on the right lists the applied CSS rules:

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

h1, h2 {
    color : gray;
}

h2 {
    color : red;
    border-bottom : 1px solid black;
}

p {
    color : blue;
}

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

The 'Computed' panel shows the final computed styles for the highlighted element, which includes the 'font-family: serif' rule from its own style block, overriding the inherited 'font-family: sans-serif' rule.

Herança, override, ... confusão!

- Como saber quais propriedades admitem herança e quais não?
 - É aí que uma **boa referência** entra em cena ([CSS Pocket Reference](#))
- Entretanto, no geral, estilos que afetam texto (e.g. `color`, `font-family`, `font-size`, `font-weight`, `font-style`) admitem herança
 - Teste `font-weight` e `font-style` e veja isso na prática!
- Outras propriedades (e.g. `border`) não serão herdadas!
 - Pense um pouco: ora, não é porque se quer colocar uma borda ao redor do corpo `<body>` do documento HTML que todos os elementos filhos (e.g. `<p>`, `<h1>`, `<a>`, ``, etc.) devem ter uma também, certo?

Obs: acima, note que, em alguns casos, vale também o bom senso na hora de se construir a regra... 23



Classes

- Com CSS, também é possível definir uma **classe** (ou grupo), à qual podem ser associados vários elementos HTML da página
 - As classes podem ser aplicadas a qualquer elemento HTML.
 - Pode-se aplicar estilos distintos para diferentes tipos de elementos do HTML, usando **classes distintas** para cada um deles
- Use o atributo **class** para adicionar elementos a uma classe
 - Um elemento pode, ainda, se relacionar com várias classes
 - Desde que os múltiplos nomes de classes no atributo **class** sejam separados com espaços em branco (mais sobre isto adiante...)

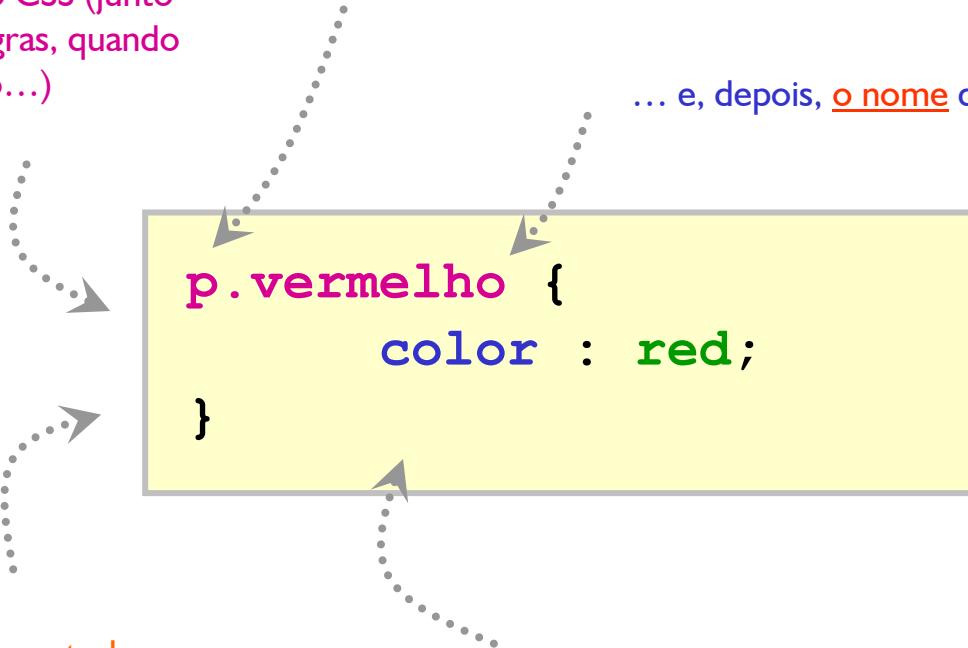


Declaração de uma classe

A definição de classes ocorre dentro do arquivo CSS (junto com as demais regras, quando for o caso...)

Uma forma simples é escrever o seletor primeiro...

... e, depois, o nome da classe.



Esta regra irá selecionar todos os parágrafos que pertençam à classe de nome “vermelho”

Use um “.” entre o nome do elemento HTML desejado e o nome da classe para selecionar um elemento específico da mesma



Relacionando-se com classes

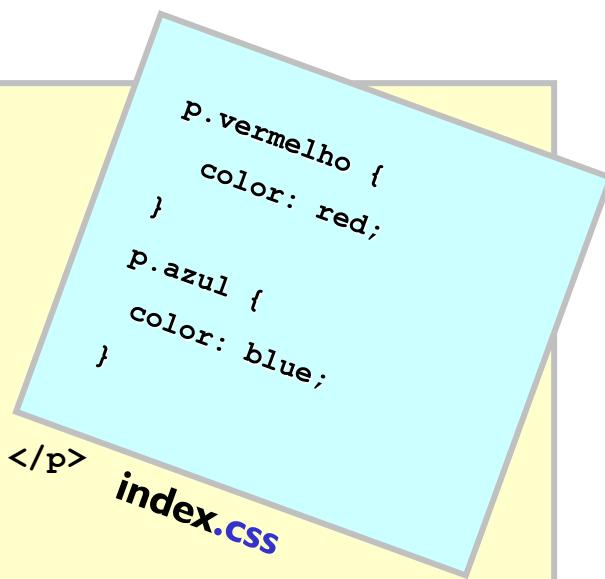
- Para associar um elemento HTML a uma classe, basta utilizar o atributo **class** (isto fará com que ele seja adicionado ao “grupo”)

Exemplo:

```
<!doctype html>

<html lang="pt">
  <head>
    <meta charset="utf-8">
    <title>O título do documento</title>
  </head>

  <body>
    <p class="vermelho"> Este parágrafo terá cor vermelha </p>
    <p class="azul"> Este parágrafo terá cor azul </p>
    ...
  </body>
</html>
```



Observe o uso do atributo e o nome da classe

Pergunta 01: e se quiséssemos criar uma classe para elementos distintos (não apenas parágrafos)?

Pergunta 02: e se fosse preciso fatorar código comum às classes (e.g., p.azul e h1.azul) ?



Classes: conceitos avançados

- É possível associar elementos distintos a uma ou várias classe(s)
 - **Vários elementos, uma classe**: se não houver nada antes do sinal de “.” e o nome da classe, todos os elementos daquela classe serão afetados

```
<h1 class="centro"> Um texto </h1>
<p class="centro"> Um texto </p>
```

```
.centro {
    text-align:center;
}
```

- **Um elemento, várias classes**: o processo inverso também é possível - um elemento pode pertencer a vários grupos simultaneamente*

```
<p class="verde borda centro">...
```

```
.borda { border-bottom: 1px solid black; }
.verde { color: green; }
.centro{ text-align:center; }
```

* E se houver regras conflitantes entre as várias classes acima? Como o browser irá decidir?



Um exemplo breve...

Laboratório 08: overriding, herança, ...



Validador CSS

- Escrever um CSS otimizado, muitas vezes, não é sinônimo de que ele foi realmente escrito sem erros de sintaxe
 - Há alguma forma de verificar se o webdesigner não cometeu erros no CSS?
- O Validador CSS da W3C
 - Da mesma forma que no validador HTML, é um serviço gratuito, online e que permite checar se um determinado CSS foi escrito corretamente
 - Por conseguinte, ele irá garantir que o CSS é bem estruturado e que todas as regras, propriedades e valores aderem a um padrão definido
- A url do validador CSS: <http://jigsaw.w3.org/css-validator/>

Uso do validador CSS

 **CSS Validation Service**
Check Cascading Style Sheets (CSS) and (X)HTML documents with style sheets

[By URI](#) [By file upload](#) [By direct input](#)

Validate by direct input

Enter the CSS you would like validated :

```
body {  
    font-family: sans-serif;  
}  
  
h1, h2 {  
    color : gray;  
}  
  
h2 {  
    color : red;  
    border-bottom : 1px solid black;  
}
```

More Options

Check



Uso do validador CSS(cont.)

 CSS Validation Service
W3C CSS Validator results for file://localhost/TextArea (CSS level 2.1)

Jump to: [Validated CSS](#)

W3C CSS Validator results for [file://localhost/TextArea](#) (CSS level 2.1)

Congratulations! No Error Found.

This document validates as [CSS level 2.1](#) !

To show your readers that you've taken the care to create an interoperable Web page, you may display this icon on any page that validates. use to add this icon to your Web page:



```
<p>
    <a href="http://jigsaw.w3.org/css-validator/">
        
    </a>
</p>
```



E a W3C ainda lhe dá a oportunidade de inserir uma imagem no document HTML indicando que a folha de estilos usada para estilizar a página é livre de erros e adere a padrões ☺!



Fontes

- É possível selecionar um trecho de texto e configurar seu tipo de fonte, tamanho, cor e até algumas “decorações” para o mesmo
- No CSS, fontes se apresentam e são classificadas em **famílias**
 - A partir daí, é só especificar as fontes que serão aplicadas em cada elemento
- Especifique o **tipo da fonte** utilizando a propriedade **font-family**

```
body {  
    font-family : Verdana, Geneva, Arial, sans-serif;  
}
```

- Apenas algumas fontes são comumente instaladas na maioria dos computadores; logo, tome cuidado na hora de escolher suas fontes!

Fontes(cont.)

- Conforme foi dito anteriormente, cada fonte pertence a uma família
 - Cada família possui um conjunto de características a serem compartilhadas
- No **CSS**, há cinco famílias básicas de fontes:

sans-serif: Verdana, Trebuchet MS, Arial, Geneva, **Arial Black**, ...

serif: Times, Times New Roman, Georgia, ...

monospace: Courier, Courier New, **Andale Mono**, ...

cursive: Comic Sans, *Apple Chancery*, ...

fantasy: **Impact**, etc...

Importante: Geneva, Courier, Helvetica e Times são fontes comumente presentes em Mac's. As demais, em plataformas Windows e Mac's (provavelmente).



Fontes(cont.)

Note o uso da propriedade font-family (e tão font-type)

A fonte Verdana está disponível no computador? Se estiver, aplique a fonte neste elemento (no caso, body)

Se Verdana não estiver disponível, então procure pela fonte Geneva. Se esta não estiver disponível, procure, então, por Arial. E assim, por diante...

```
body {  
    font-family : Verdana, Geneva, Arial, sans-serif;  
}
```

Não há uma regra de quantas fontes devem aparecer aqui. Porém, é sempre bom colocar o nome de uma família como uma última opção de contingência para aqueles que não possuem uma fonte específica

Esta lista de prioridades deve ser separada por vírgulas, a partir da fonte mais específica até a mais genérica. Se houver a necessidade de especificar uma fonte de nome composto, use aspas duplas (e.g. "Times New Roman")

Finalmente, se nenhuma das fontes for encontrada, o browser irá aplicar o que ele considerar “default” para a fonte sans-serif

Importante: os nomes das fontes são case-sensitive.



Fontes(cont.)

- Altere a intensidade (peso) da fonte com a propriedade **font-weight**

```
h1 {  
    font-weight : bold;  
}
```

O peso pode variar ligeiramente de um navegador para outro.

lighter *

normal

bold

bolder

- Especifique a “decoração” com a propriedade **text-decoration**

```
p {  
    text-decoration : underline;  
}
```

É possível tornar um texto decorado usando-se também outros valores de propriedades:

none

underline

overline

line-through

blink *

* Ainda não suportada por todos os navegadores.



Fontes(cont.)

- Especifique o **tamanho** da fonte com a propriedade **font-size** e sua cor usando a propriedade **color**

```
body {  
    font-size : 16px;  
    color : red;  
}
```

Também é possível especificar o tamanho da fonte utilizando outras unidades além do pixel. E a cor utilizando a notação Hexadecimal. Mais sobre estes dois pontos adiante...

- Veja algumas cores pré-definidas:

aqua		green		navy		silver	
blue		gray		olive		teal	
black		lime		purple		white	
fuchsia		maroon		red		yellow	



Fontes(cont.)

- É possível ajustar o tamanho do texto usando pixels, conforme visto
- Porém, também é possível ajustar o tamanho da fonte de forma **relativa** a uma outra fonte (pai ou a folha de estilos do browser)
 - Isto é feito com duas unidades relativas: **%** e **em**

É facil: defina um tamanho para o <body>.
Isto irá agir como um default para a página

Uma boa prática: agora, defina os tamanhos de fontes dos outros elementos de forma relativa ao tamanho do corpo (com % ou em). Assim, se o tamanho das fontes tiver de mudar, ajustes serão realizados em apenas um único local (além disso, ajustes por parte dos usuários também serão permitidos) ☺. Pense nisso...

```
body { font-size : 14px; }
h1 { font-size : 150%; }
h2 { font-size : 1.2em; }
```

Importante: não confunda “em” com !



Fontes(cont.)

- Infelizmente, algumas versões de navegadores (e.g. Internet Explorer) não permitem, aos visitantes da página, ajustar o tamanho do texto quando o seu valor-base é expresso em pixels
 - Para evitar isso, use **keywords** ao invés de pixels no valor-base

```
body {  
    font-size : small;  
}  
  
h1 {  
    font-size : 150%;  
}  
  
h2 {  
    font-size : 1.2em;  
}
```



xx-small
x-small
small
medium
large
x-large
xx-large

Os valores reais das keywords não são constantes entre todos os navegadores!



Fontes(cont.)

- Adicione **ítálico** a um elemento com a propriedade **font-style**

```
p {  
    font-style : italic;  
}
```

Um texto normal

Um texto em itálico

- Entretanto, nem todas as fontes fornecem suporte nativo a itálico

- Se isso acontecer, use o valor **oblique**

```
p {  
    font-style : oblique;  
}
```

...mas, por que não usar `` em vez de todo este cuidado?

oblique, na verdade, também indica ao navegador que um texto “em itálico” é desejado. Porém, ao invés de procurar e usar uma fonte projetada especialmente para este fim, o browser irá aplicar uma “inclinação” no texto “na marra”.

- Pergunta intrigante:

Na prática, entretanto, dependendo da escolha da **fonte** e do seu **browser**, os dois estilos poderão ser idênticos.

```
h1 { text-transform: uppercase; }
```

...e `text-transform`? O que vai acontecer?

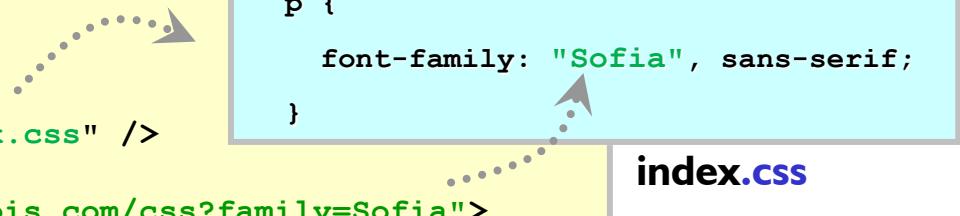


Referenciando fontes

- Indique a fonte desejada usando a API do Google Fonts:

```
<!doctype html>
<html lang="pt">
  <head>
    ...
    <link rel="stylesheet" href="index.css" />
    <link rel="stylesheet"
          href="https://fonts.googleapis.com/css?family=Sofia">
  </head>
  <body>
    ...

```



- Ou référencia o arquivo da fonte locamente, com **@font-face**

Baixe em: <https://fonts.google.com/specimen/Lobster>

```
@font-face{
  font-family: Lobster;
  src: url('../fonts/Lobster-Regular.ttf')
}
```

Neste caso, faça download do arquivo
da fonte e indique sua localização

```
pasta-raiz-site
├── css
│   └── index.css
└── fonts
    └── Lobster-Regular.ttf
    └── index.html
```



Cores

- Já foi visto como especificar cores usando nomes de cores
 - A paleta CSS de nomes contém pouco mais de 10 cores (fator limitante)
- Porém, não seria mais conveniente especificar cores usando uma “paleta” de aproximadamente 16 milhões de cores? Como?
 - Porcentagens de Red Green Blue, (usando valores de 0-255, ou %)
 - Notação Hexadecimal (seis dígitos, de 0..9, A, B, C, D, E e F)
- Uma maneira interessante de mixar cores é utilizando ferramentas de edição de fotos ou até mesmo usando alguns catálogos da Web

Cores(cont.)

- Crie uma cor em termos de valores **RGB** (mixagem de cores)

```
h1 {  
    background-color : rgb(80%, 40%, 0%);  
}
```

80% de vermelho, 40% de verde e 0% de azul...

- Ou até mesmo especifique em termos de valores finais, diretamente

```
h1 {  
    background-color : rgb(204, 102, 0);  
}
```

80% de 255 = 204

40% de 255 = 102

00% de 255 = 0

```
h1 {  
    background-color : rgba(204, 102, 0, 0.61);  
}
```

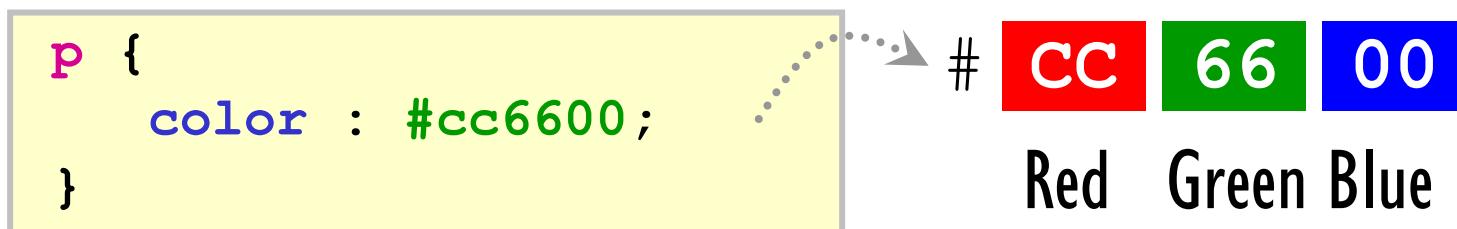
alpha = %transparência

Obs: veja que é difícil “adivinar” a cor sem o auxílio de uma ferramenta



Cores(cont.)

- Você também pode “misturar” cores usando a notação Hexadecimal
 - E, nesta notação, os dígitos para as cores podem conter letras!
- Na verdade, o primeiro conjunto de dois dígitos representa o **vermelho**, outro conjunto o **verde** e os restante para o **azul**
 - Sempre inicie uma notação hexa com o sinal “#”



- Onde:

CC
66
00

$$12(\text{C}) * 16 + 12 = 192 + 12 = 204$$

$$6 * 16 + 6 = 96 + 6 = 102$$

$$0 * 16 + 0 = 0 + 0 = 0$$



Catálogo (uma parte...) de cores na Web

HTML name	Hex code R G B	Decimal code R G B	HTML name	Hex code R G B	Decimal code R G B	HTML name	Hex code R G B	Decimal code R G B
Red colors			Green colors			Brown colors		
IndianRed	CD 5C 5C	205 92 92	GreenYellow	AD FF 2F	173 255 47	Cornsilk	FF F8 DC	255 248 220
LightCoral	F0 80 80	240 128 128	Chartreuse	7F FF 00	127 255 0	BlanchedAlmond	FF EB CD	255 235 205
Salmon	FA 80 72	250 128 114	LawnGreen	7C FC 00	124 252 0	Bisque	FF E4 C4	255 228 196
DarkSalmon	E9 96 7A	233 150 122	Lime	00 FF 00	0 255 0	NavajoWhite	FF DE AD	255 222 173
LightSalmon	FF A0 7A	255 160 122	LimeGreen	32 CD 32	50 205 50	Wheat	F5 DE B3	245 222 179
Crimson	DC 14 3C	220 20 60	PaleGreen	98 FB 98	152 251 152	BurlyWood	DE B8 87	222 184 135
Red	FF 00 00	255 0 0	LightGreen	90 EE 90	144 238 144	Tan	D2 B4 8C	210 180 140
FireBrick	B2 22 22	178 34 34	MediumSpringGreen	00 FA 9A	0 250 154	RosyBrown	BC 8F 8F	188 143 143
DarkRed	8B 00 00	139 0 0	SpringGreen	00 FF 7F	0 255 127	SandyBrown	F4 A4 60	244 164 96
Pink colors			MediumSeaGreen	3C B3 71	60 179 113	Goldenrod	DA A5 20	218 165 32
Pink	FF C0 CB	255 192 203	SeaGreen	2E 8B 57	46 139 87	DarkGoldenrod	B8 86 0B	184 134 11
LightPink	FF B6 C1	255 182 193	ForestGreen	22 8B 22	34 139 34	Peru	CD 85 3F	205 133 63
HotPink	FF 69 B4	255 105 180	Green	00 80 00	0 128 0	Chocolate	D2 69 1E	210 105 30
DeepPink	FF 14 93	255 20 147	DarkGreen	00 64 00	0 100 0	SaddleBrown	8B 45 13	139 69 19

Disponível por HTTP em: http://en.wikipedia.org/wiki/Web_colors



Um exemplo breve...

Laboratório 09: validador CSS, fontes, cores, ...



Box Model

- **CSS** permite utilizar um modelo de caixa para controlar como elementos (tag's) html serão exibidos no navegador
 - As “caixas” têm um conteúdo, um padding*, uma borda* e uma margem*
- Cada uma destas partes são comentadas abaixo:
 - **Conteúdo**: a área alocada para o conteúdo consiste do espaço mínimo necessário para receber o conteúdo (até tocar a borda do browser, se for o caso)
 - **Padding**: propriedade utilizada para criar um espaço ao redor do conteúdo (espaço interno entre o conteúdo e a borda, quando houver uma, e é parte do elemento)
 - **Borda**: fica ao redor do padding (se houver algum) e, por sua vez, do conteúdo
 - **Margem**: fica fora da borda, não é parte do elemento; é só usada para fornecer espaçamento entre elementos adjacentes*

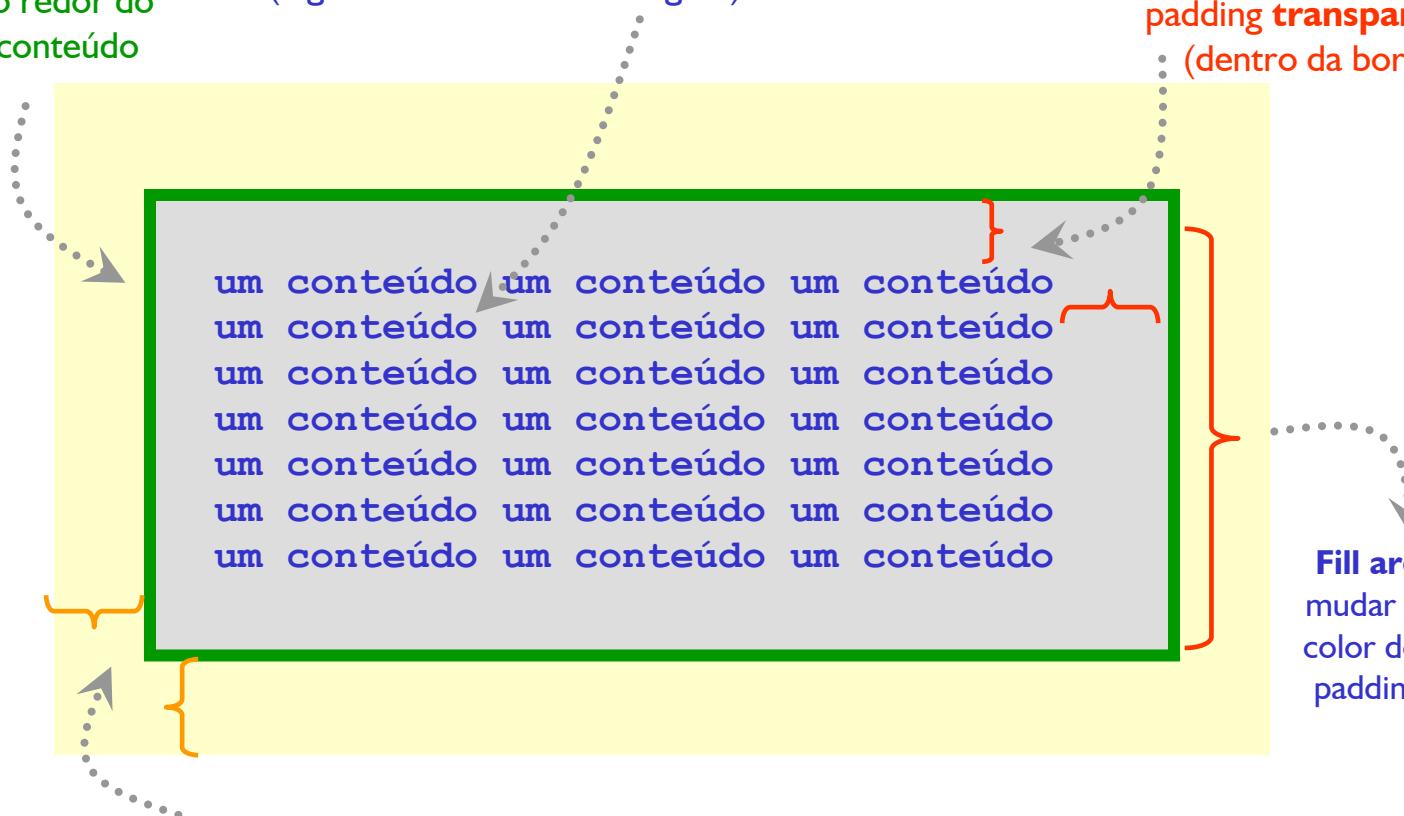
* padding, borda e margem são, conceitualmente, opcionais! Não esqueça disso...

Box Model(cont.)

Uma borda (opcional) pode ser colocada ao redor do padding e do conteúdo

Esta parte é chamada de conteúdo
(e.g. um texto ou uma imagem)

Ao redor do conteúdo, pode (não é obrigatório) haver um padding transparente (dentro da borda)



Finalmente, uma margem transparente e opcional pode ser colocada ao redor de todo o resto (fora da borda e fora do elemento)

Todos os elementos são tratados como caixas: parágrafos, cabeçalhos, listas, itens de lista, imagens, texto enfatizado, links, quotas, citações em bloco, etc



Pode-se gerar várias combinações...

um conteúdo um conteúdo
só conteúdo

um conteúdo um conteúdo
um conteúdo um conteúdo
um conteúdo um conteúdo
um conteúdo um conteúdo
um conteúdo um conteúdo
um conteúdo um conteúdo
um conteúdo um conteúdo
um conteúdo um conteúdo
um conteúdo um conteúdo
um conteúdo um conteúdo
um conteúdo um conteúdo
só bordas

um conteúdo um conteúdo
um conteúdo um conteúdo

só margens

um conteúdo um conteúdo um conteúdo
um conteúdo um conteúdo um conteúdo

padding iguais

padding sup. e esq. diferentes

um conteúdo um conteúdo
um conteúdo um conteúdo

um conteúdo um conteúdo um conteúdo
um conteúdo um conteúdo um conteúdo
um conteúdo um conteúdo um conteúdo
um conteúdo um conteúdo um conteúdo
um conteúdo um conteúdo um conteúdo
um conteúdo um conteúdo um conteúdo
um conteúdo um conteúdo um conteúdo

só margem dir. + padding sup.inf.

um conteúdo um conteúdo
margem sup. e inf. + borda

um conteúdo um conteúdo
um conteúdo um conteúdo

margens inf. e dir. distintas + borda esq.

misturado 😊

um conteúdo um conteúdo
um conteúdo um conteúdo
um conteúdo um conteúdo



Bordas

- Altere a **espessura** da borda com a propriedade **border-width**

```
p {  
    border-width : thin;  
}  
  
img {  
    border-width : 5px;  
}
```

thin _____

medium _____

thick _____

1px	
2px	
3px	
4px	
5px	
6px	

Você pode especificar usando thin, medium, thick ou usando px

- Altere a **cor** da borda com a propriedade **border-color**

```
p {  
    border-color : red;  
    border-color : rgb(100%, 0%, 0%);  
    border-color : #FF0000;  
}
```

Você pode usar cores pré-definidas, valores em rgb ou usando a notação hexadecimal

HTML name	Hex code R G B	Decimal code R G B	HTML name	Hex code R G B	Decimal code R G B	HTML name	Hex code R G B	Decimal code R G B
Red colors								
IndianRed	CD 5C 5C	205 92 92	GreenYellow	AD FF 2F	173 255 47	CornSilk	FF F8 DC	255 248 220
LightCoral	F0 80 80	240 128 128	Chartreuse	7F FF 00	127 255 0	BlanchedAlmond	FF EB CD	255 235 205
Salmon	FA 80 72	250 128 114	LawnGreen	7C FC 00	124 252 0	Bisque	FF E4 C4	255 228 196
DarkSalmon	E9 96 7A	233 150 122	Lime	00 FF 00	0 255 0	NavajoWhite	FF DE AD	255 222 173
LightSalmon	FF A0 7A	255 160 122	LimeGreen	32 CD 32	50 205 50	Wheat	F5 DE B3	245 222 175
Crimson	DC 14 3C	220 20 60	PaleGreen	98 FB 98	152 251 152	BurlyWood	DE B8 87	222 184 135
Red	FF 00 00	255 0 0	LightGreen	90 EE 90	144 238 144	Tan	D2 B4 8C	210 180 140
FireBrick	B2 22 22	178 34 34	MediumSpringGreen	00 FA 9A	0 250 154	RosyBrown	BC 8F 8F	188 143 143
DarkRed	8B 00 00	139 0 0	SpringGreen	00 FF 7F	0 255 127	SandyBrown	F4 A4 60	244 164 96
Pink colors								
Pink	FF CO CB	255 192 203	MediumSeaGreen	3C B3 71	60 179 113	Goldenrod	DA A5 20	218 165 32
LightPink	FF B6 C1	255 182 193	SeaGreen	2E 8B 57	46 139 87	DarkGoldenrod	B8 86 0B	184 134 11
HotPink	FF 69 B4	255 105 180	ForestGreen	22 88 22	34 139 143	Peru	CD 85 3F	205 133 63
DeepPink	FF 14 93	255 20 147	Green	00 80 80	0 128 0	Chocolate	DD 69 21	210 105 30
						SaddleBrown	8B 45 13	139 69 19

Vide catálogo ...



Bordas(cont.)

- Altere o estilo da borda com a propriedade **border-style**

```
h1 {  
    border-style : dotted;  
}
```

Explore cada um dos valores (de preferência, no Chrome e Firefox...) !



solid	dotted*
double	dashed
groove*	inset
outset	ridge*

- Especificando os lados da borda:

border-top-color
border-top-style
border-top-width

border-right-color
border-right-style
border-right-width

border-bottom-color
border-bottom-style
border-bottom-width

border-left-color
border-left-style
border-left-width

Exemplo:

```
h1 {  
    border-top-style : solid;  
    border-top-color : red;  
    border-top-width : 2px;  
}
```

Atalho: **border: 2px solid red;**

* Pode não ser suportada em navegadores mais antigos.

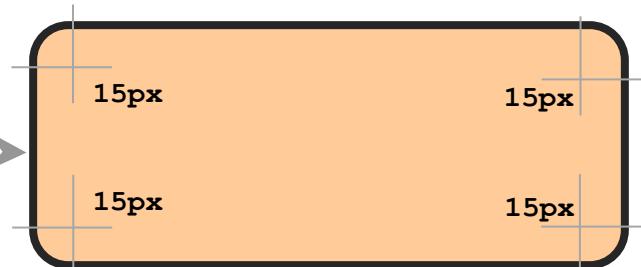


Bordas(cont.)

- Pode-se “arredondar” todas as bordas, com **border-radius**

```
h1 {  
    border-radius : 15px;  
}
```

Explore cada um dos valores (de preferência, no Chrome e Firefox...) !



- Ou definir cada um dos cantos da borda isoladamente:

border-top-left-radius

border-top-right-radius

border-bottom-left-radius

border-bottom-right-radius

Exemplo:

```
h1 {  
    border-top-left-radius:15px;  
    border-top-right-radius:15px;  
    border-bottom-left-radius:15px;  
    border-bottom-right-radius:15px  
}
```

* Pergunta: ao lado, quais propriedades devo usar?.....?



(margem e bordas)

- Pode-se usar **atalhos** para diminuir a escrita no arquivo CSS

```
div {  
    padding-top : 0px;  
    padding-right : 20px;  
    padding-bottom : 30px;  
    padding-left : 10px;  
}
```

Mais simples!

```
padding: 0px 20px 30px 10px;
```

- Também podem ser usados em margens:

```
div {  
    margin-top : 0px;  
    margin-right : 20px;  
    margin-bottom : 0px;  
    margin-left : 20px;  
}
```

Ou ainda mais simples!

```
margin: 0px 20px;
```

* Acima: top & bottom = 0px ; right & left = 20px

Explore cada um dos atalhos mostrados!



Global Reset Rule

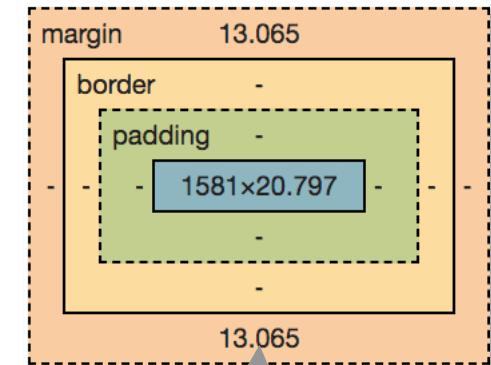
- Muitas vezes, ao usar o Chrome Dev Tools para inspecionar a **margin**, pode-se ver algum valor default do navegador
 - Mas que não foi configurado pelo Web Designer!
- Neste caso, uma solução bastante comum é iniciar o arquivo.css com uma regra de **global reset**

arquivo.css

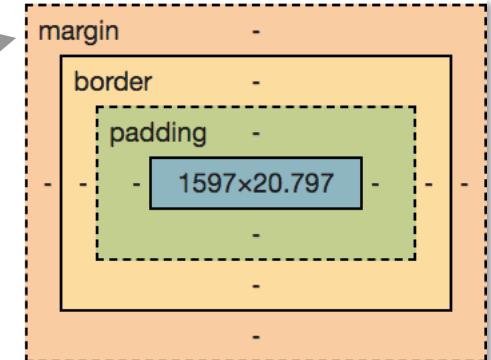
```
* {  
    margin : 0;  
    padding : 0;  
} /* global reset */
```

A global reset irá
remover qualquer
configuração de
margem e padding

- E depois override da **margin** nas regras seguintes



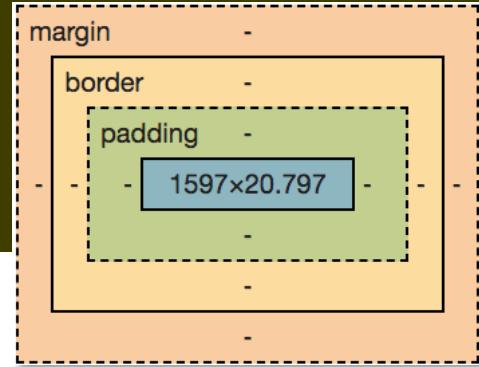
O que é isso?



Atenção: para dar espaçamento vertical entre elementos **block**, se usar margin top & bottom ao mesmo tempo, cuidado no **colapso/overlap** de margens superior e inferior. Apenas a maior margem final será aplicada. Vide https://www.w3schools.com/css/css_marginCollapse.asp



Recapitulando...



- Elemento é tratado pelo browser como uma caixa
 - O elemento tem seu conteúdo, padding, borda e margem
 - O **padding** é “parte do elemento” e representa o espaço interno entre o conteúdo e borda; ou seja, é o espaço “dentro” do elemento e ao redor do conteúdo;
 - Para criar um espaço fora do elemento, após a borda, ou até para adicionar espaçamento entre os vários elementos adjacentes, use **margin**
 - **Atenção:** padding é afetado pela background-color do elemento, mas não as margens
 - Caso precise acrescentar espaço vertical entre elementos, a dica é tentar padronizar e usar ou **margin-top** ou **margin-bottom** para realizar a tarefa *
 - Use a global reset rule para zerar algum valor de margin/padding desconhecido
 - Se não padronizar e aplicar os espaçamentos verticais misturando **margin-top** e **margin-bottom**, posso ter resultados estranhos, como o colapso/overlap das margens inferior e superior

***Atenção:** cuidado no colapso/overlap de margens superior e inferior.
https://www.w3schools.com/css/css_marginCollapse.asp



***Atenção:** não haverá colapso/overlap de margens horizontais em inline elements.

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flow_layout/Block_and_inline_layout_in_normal_flow

Um exemplo breve...

Laboratório 10: bordas, box model, ...



O atributo id

- Até aqui, temos usado classes quando desejávamos aplicar um mesmo estilo a um grupo específico de elementos HTML da página
 - Mas, e quando este grupo contém apenas **um único elemento** (e.g., o rodapé da página, ou um parágrafo contendo uma mensagem do dia, ...)?
- Se **há apenas um único elemento** a ser estilizado de maneira diferente dos demais, classes não são (semanticamente) a melhor solução!
- Use o atributo **id** para prover a um elemento um nome único
 - Deve haver apenas um único elemento na página com um dado **id**
 - A partir daí, pode-se usar este **id** para prover um único estilo ao elemento

Anatomia de um id

A definição de um id ocorre dentro do arquivo CSS (junto com as demais regras, quando for o caso...)

Uma forma simples é escrever # primeiro...

... e, depois, o nome do id.

```
#vermelho {  
    color : red;  
}
```

Esta regra irá selecionar o elemento que possuir o atributo id de nome “vermelho”

Evite utilizar espaços em branco e/ou caracteres especiais para nomear um id

Use um “#” seguido do nome do id para selecionar um único elemento da página



id (conceitos avançados)

- E se houver dois elementos de mesmo **id** na página e uma regra id?
 - Ambos serão estilizados, mas, seu documento HTML não será validado!
- Um elemento pode ter um **id** único e ainda fazer parte de uma classe
 - Mas... e se houver regras conflitantes escritas nas regras da classe e do id?
 - Uma regra escrita com id deterá maior prioridade em relação a uma regra escrita com elementos HTML ou até mesmo classes! Pense nisso...
- Vale também saber que um **id** pode ser utilizado como um “alvo” na criação de links internos (vide módulo HTML) *
 - Isso já é uma feature da W3C (tornou as coisas mais simples...)

* Isso não era suportado todos os navegadores (sobretudo os mais antigos).



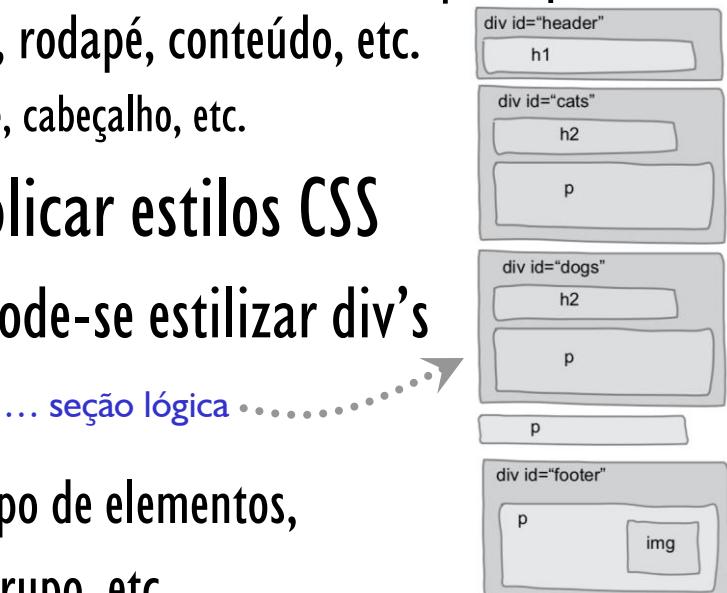
Quiz...

- Qual a escolha mais adequada em cada item abaixo? **id** ou **class**?
 - Um parágrafo contendo o rodapé da página html
 - Uma imagem na página representando a “foto do dia”
 - Parágrafos que contém comentários de visitantes sobre um dado filme
 - A lista de produtos em um carrinho de compras (a imagem do produto, sua descrição, preço, quantidade em estoque, etc.)
 - O parágrafo que foi estilizado na prática 10?
 - A barra de navegação superior em um site (contendo menu de links)



Elemento <div>

- Elementos <div> são usados, basicamente, para agrupar elementos relacionados na página de modo a criar **seções lógicas**
 - Na verdade, ele age como um “contêiner” para vários outros elementos HTML que devam ser relacionados dentro da página
 - Elementos <div> ajudam o webdesigner a identificar e delimitar as principais “grandes áreas” de um documento: cabeçalho, rodapé, conteúdo, etc.
 - No HTML5, novas tags foram criadas para rodapé, cabeçalho, etc.
- <div> pode agrupar elementos para aplicar estilos CSS
 - Uma vez que elementos foram agrupados, pode-se estilizar div's como outros elementos HTML quaisquer.
 - Exemplo: criar uma borda ao redor de um grupo de elementos, adicionar uma mesma cor de background ao grupo, etc.



* Fonte: Head First CSS HTML



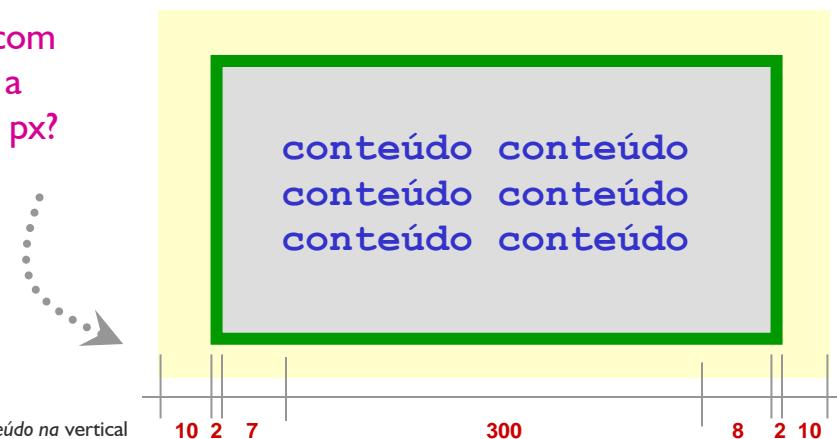
Propriedade width

- É possível usar a propriedade **width** para configurar a largura de elementos
 - Se não houver uma regra alterando a largura de um elemento, o browser, então, irá admitir a largura como sendo “auto”...
 - E irá expandir o elemento usando todo o espaço disponível (na área visível do browser)
- Por padrão, a propriedade **width** especifica somente a largura do **conteúdo**
 - Assim, se for preciso descobrir a largura total de um elemento, deve-se somar possíveis valores de padding's, bordas e margens (quando for o caso)

Ex.:

À direita, encontra-se um parágrafo com algumas indicações CSS. Qual será a largura total do elemento na tela, em px?

```
p {  
    border-width : 2px;  
    padding: 5px 8px 10px 7px;  
    margin: 10px;  
    width: 300px; /*content only*/  
}
```



* Geralmente, seta-se o width e deixa o height no default: auto; para evitar overflow de conteúdo na vertical

* Se usar porcentagem, width será calculado a partir do element parent (pode ser body, div, etc.)

Propriedade width (dica!)

- Porém, um truque é usar a propriedade **box-sizing** para modificar este comportamento default do Box Model
 - Com **box-sizing**, as propriedades **width** e **height** irão incluir, na largura/altura do elemento, seu **conteúdo**, **bordas** e **paddings** (só não incluirá as margens!)
 - Eventuais valores de paddings e bordas estarão “dentro” do valor da **width/height** da box
 - Não é preciso mais somar valores de padding's e bordas para descobrir largura/altura total

À direita, nos valores de **width** e **height**, já estarão incluídos os eventuais valores de paddings, bordas e o conteúdo !

$$\text{width} = \text{border} + \text{padding} + \text{content-width}$$
$$\text{height} = \text{border} + \text{padding} + \text{content-height}$$

Ex.:

Cuidado! **border-box** não incluirá valores de margens!

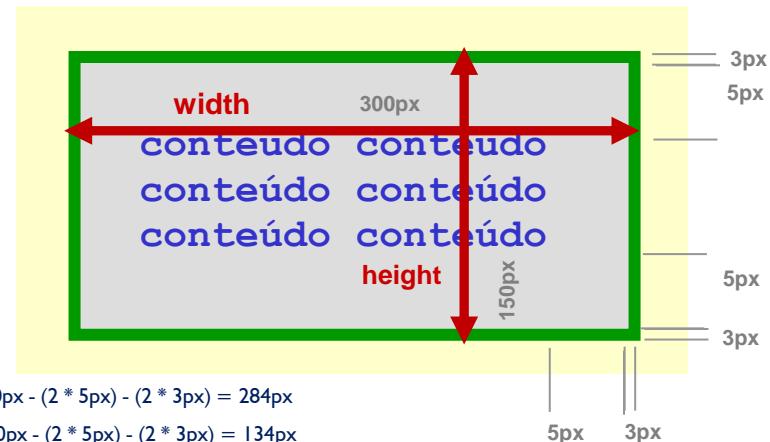
```
p {  
    border-width: 3px solid green;  
    padding: 5px;  
    margin: 10px; /* não conta */  
    width: 300px;  
    height: 150px;  
    border-sizing: border-box;  
} /* pode colocar na global-reset rule */
```

Total width: 300px

Total height: 150px

Content box width: $300px - (2 * 5px) - (2 * 3px) = 284px$

Content box height: $150px - (2 * 5px) - (2 * 3px) = 134px$



Propriedade width (em imagens)

- Alguns consideram omitir as propriedades **width** e **height** para configurar as dimensões de uma imagem dentro do arquivo HTML uma boa prática

```
>
```

Pois, como as dimensões estão no HTML em px, o *aspect ratio* será “esticado” se eu mudar os valores e deixá-los diferentes dos valores originais da dimensão da imagem

- Porém, se não especificar largura e altura no html, o browser não poderá alocar o espaço
 - E não poderá calcular e reservar o espaço antes de carregar a imagem. Isto pode causar **jank**¹ indesejado. Por isso, a documentação² sugere manter width e height no html
 - Ainda, no CSS, uma forma simples de poder alterar o *aspect ratio* sem grandes distorções da imagem é manter **width** e setar **height** como auto (ou vice-versa)

```
img {  
  width: 300px;  
  height: auto;  
}  
arquivo.css
```

Se eu fizer override do width da imagem no css, o height será recalculado automaticamente pelo browser, para evitar distorções

... e se setar width assim?

```
img {  
  width: 100%;  
  height: auto;  
}  
arquivo.css
```

¹ <https://developer.mozilla.org/en-US/docs/Glossary/Jank>

² https://developer.mozilla.org/en-US/docs/Learn/Performance/Multimedia#rendering_strategy_preventing_jank_when_loading_images

* Se usar %, width é calculado em termos do parent element

Centralizar página com width (dica)

- É possível centralizar o conteúdo de toda a página com a propriedade **width**
- Nas páginas construídas até aqui, o browser sempre redimensionava o conteúdo dos elementos (a ideia de um “**fluid**” layout), expandindo-os de um lado para outro
 - Você podia testar isso redimensionando a janela do browser — percebia-se que o conteúdo dos elementos eram expandidos e ajustados até o limite da janela do navegador
- E como fazer um “**frozen**” layout com a página sempre centralizada? Um truque simples é:
 - colocar todos os elementos da página em um **contêiner** (e.g., um **div**); este contêiner deve ter sua propriedade **width** configurada; por exemplo, **width=750px**
 - configurar ambas as margens esquerda e direta com auto

arquivo.css

```
div#main-page {  
    width : 750px;  
    margin-left : auto;  
    margin-right : auto;  
} /* centralizar */
```

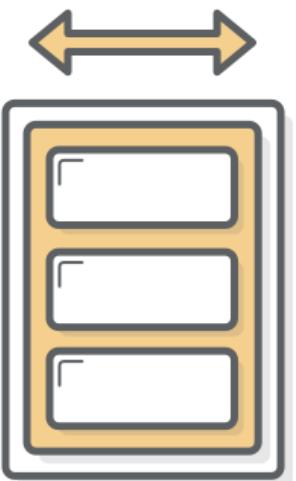
width irá congelar a largura de todos os filhos do contêiner

auto significa que o browser irá recalcular e aplicar os valores das margens automaticamente, sempre que a janela for redimensionada

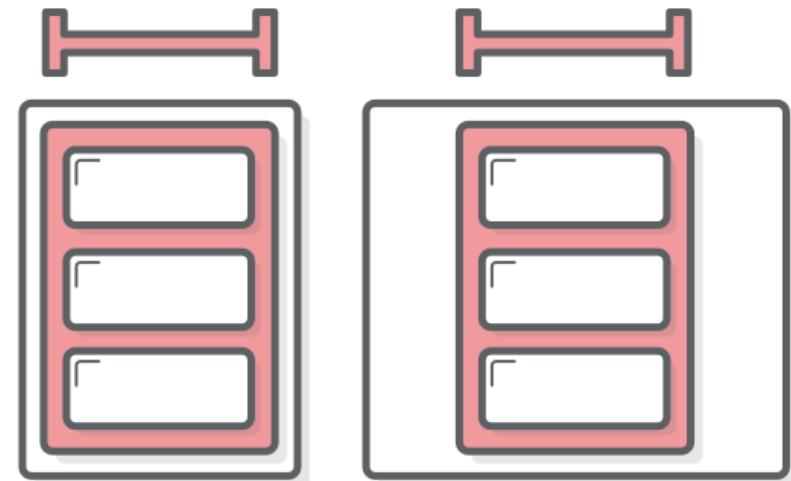
```
<!doctype html>  
  
<html lang="pt">  
    <head> ... </head>  
    <body>  
        <div id="main-page">  
            <p> ... </p>  
            <h1> ... </h1>  
            <img src ... >  
            ...  
        </div>  
    </body>  
</html>
```

arquivo.html

Centralizar a página com width (cont.)



FLUID LAYOUT



FIXED-WIDTH LAYOUT



Um exemplo breve...

Laboratório II: uso de <div> prática lounge do
livro Head First HTML & CSS, 2nd Edition



Elementos descendentes

- Com CSS, também é possível selecionar apenas determinados elementos descendentes (elementos filhos, netos, bisnetos, etc.)

Selecione apenas os parágrafos que são descendentes (em qualquer nível) da div rodapé

```
div#rodape p {  
    color : #FF4567;  
}
```

- Pode-se fazer construções mais complexas...

Selecione apenas os parágrafos que são filhos (child) diretos da div. Se houvesse, por ex., um blockquote na div que possuísse um parágrafo como filho, este não iria ser afetado, pois já seria, no caso, um “neto”...

```
div#cabeçalho > p {  
    color : #FF4567;  
}
```

Selecione apenas aqueles parágrafos que aparecem imediatamente após os elementos h1

```
h1+p {  
    color : #FF4567;  
}
```



Propriedade text-align

- A propriedade `text-align` permite alinhar todos os elementos “inline” (ou apenas texto) que estão dentro de elementos de bloco
 - Lembre-se de que, apesar de `text-align` funcionar para qualquer elemento inline, ela sempre deve ser setada em elementos de bloco!
 - Logo, tentar usar diretamente `text-align` em `` não irá funcionar
 - Mas, se `` estiver dentro de uma `<div>`, ela será alinhada
- Note que é bastante comum setar `text-align` em um elemento “pai” e o alinhamento ser passado para os filhos (inline’s ou blocos)
 - Ex: `text-align` em uma `<div>` que contém `<p>` e `<h1>`
 - Estes dois últimos elementos herdarão o alinhamento da *parent* `<div>`

Elemento

- O elemento é bastante similar ao <div> visto há pouco: é utilizado para agrupar elementos “inline” (em linha) ou texto puro
 - Da mesma forma que com as div's, é possível adicionar span's à classes (ou até prover um **id** único para eles) visando estilizá-los em um arquivo CSS!

- Exemplo:

arquivo.html

```
<p> Camisas <span class="vermelho"> vermelhas </span>  
terão 30% de desconto hoje mesmo! </p>  
  
<h3> Afinal, o <span class="vermelho"> vermelho </span>  
é uma cor que está sempre na moda </h3>
```

arquivo.css

```
.vermelho {  
color: red;  
font-weight: bold;  
}
```

- Pergunta intrigante:

...mas, por que não usar no
html e no css em vez de todo este cuidado?

 é um elemento inline. Logo, ele não causará
nenhuma quebra de linha indesejada em um texto
que, supostamente, deva ser contínuo...



<a> e pseudo-classes

- O elemento <a> é um exemplo de elemento que admite diferentes **estados** (não-clicado, clicado, mouse sobre o link, etc....)
 - A folha de estilos default do browser provê uma aparência para cada estado
- É bastante comum aplicar estilos a cada um destes estados de <a>
 - Use **pseudo-classes** para contemplar cada estado possível
 - É possível estilizar pseudo-classes, mas elas não irão aparecer na estrutura HTML
 - O browser é que irá fazer todo o trabalho de associar o link à classe correta com base na interação do visitante enquanto ele usa os links (isso tudo “on-the-fly”)

Teste isso...



```
a:link { color: green; } /* link não visitado */
a:visited { color: red; } /* link já visitado */
a:hover { color: yellow; } /* mouse sobre o link */
a:active { color: blue; } /* segurar o click */
```

pseudo-classes (cont.)

- Com **pseudo-classes**, também é possível selecionar apenas determinados elementos descendentes (elementos filhos, etc.)

Selecione todos parágrafos, onde cada um seja o primeiro filho direto de um elemento qualquer

E se usasse **p:last-child** ?

E se usasse **p:first-of-type** ?

```
p:first-child {  
    font-weight : bold;  
}
```

- Outras construções complexas...

Selecione todos os segundos itens de lista filhos de elementos ou quaisquer

E se usasse **li:nth-child(odd)** ?

```
li:nth-child(2) {  
    color : green;  
}
```

O quê o seletor ao lado faz?

a) seleciona só os primeiros parágrafos das <div>'s?

b) os parágrafos descendentes de <div>'s que sejam os primeiros filhos diretos de elementos quaisquer ?

```
div p:first-child {  
    color : green;  
}
```



pseudo-classes (cont.)

- Nos trechos HTML/CSS abaixo, quais elementos serão afetados?

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

```
div p {  
    background-color : yellow;  
}
```

```
div > p {  
    color : blue;  
}
```

```
p:first-child {  
    color : green;  
}
```

```
div p:first-child {  
    color : red;  
}
```

```
...  
<body>  
  
<h2>um texto de cabeçalho</h2>  
  
<p>Parágrafo 0. Não está na div.</p>  
  
<div>  
    <p>Parágrafo 1 na div.</p>  
    <p>Parágrafo 2 na div.</p>  
    <section>  
        <p>Parágrafo 3 na div.</p>  
    </section>  
    <p>Parágrafo 4 na div.</p>  
</div>  
  
<article><p>Parágrafo 5 em article.</p></article>  
<p>Parágrafo 6. Não está na div.</p>  
  
</body>  
...
```



display[block,inline,inline-block]

- Os tipos de caixas (boxes), a saber: block, inline e inline-block
- Muitos elementos no HTML são tratados como caixas de bloco, ou block boxes
 - Tentam ocupar toda a largura disponível (ainda que conteúdo seja menor)
 - Efeito de *linebreak*; e a ideia é que block boxes são *stacked* um sobre o outro;
 - Não permitem elementos ao seu lado, por padrão;
 - Atenção: height, margin, width e padding são respeitadas;
 - No CSS, a propriedade padrão é: display:block
- O comportamento de inline boxes é bem diferente:
 - Não há *linebreaks*; e podem estar lado a lado
 - Ocupam só o espaço necessário para o conteúdo;
 - Atenção: height e width não são respeitadas;
 - padding e margin apenas se for na horizontal* (left,right)
 - No CSS, a propriedade padrão é: display:inline

The screenshot shows two code snippets and their corresponding browser rendering.

Top Snippet:

```
h1 {  
  width: 100px;  
  height: 50px;  
}  
  
h1 {  
  display: block;  
}
```

Bottom Snippet:

```
element.style {  
}  
  
strong {  
  background-color: red;  
  width: 100px; ①  
  height: 100px; ②  
  margin-top: 50px;  
  padding-top: 200px;  
  margin-left: 30px;  
}
```

The browser rendering shows a large red box with a height of 50px and a width of 100px, as defined by the first snippet. The second snippet's styling (background-color, width, height) is applied to the **element, demonstrating that inline elements ignore width and height properties.**

Ex.: <https://codepen.io/C-sar-Rocha/pen/eYQQdg>

The `display: inline` property prevents `width` from having an effect.
Try setting `display` to something other than `inline`.

Ex.: <https://codepen.io/C-sar-Rocha/pen/OjaabXg> 73



* Importante: ainda que se aplique nos 4 lados da inline box e veja os valores no DevTools, na prática, não é possível ver efeitos na renderização!

display[block,inline,inline-block](cont.)

- Em caixas do tipo inline-block, têm-se o melhor dos dois mundos:
 - Elementos são tratados como inline & block boxes (combinação)
 - Elementos só ocupam a largura necessária para o conteúdo (inline)
 - Permitem caixas lado a lado, sem efeito linebreak (inline)
 - Porém, volta a considerar as propriedades height e width
 - Além de respeitar padding e margin na vertical (block)
 - No CSS, a propriedade é: **display:inline-block**

```
.lateral-esq {  
    background-color: red;  
    display: inline-block;  
}  
  
section {  
    display: block;  
}
```

Ao lado, uma <section> que, por padrão, seria display block, agora overridden como inline-block

Ex.: <https://codepen.io/C-sar-Rocha/pen/JjeeNMD>

- Um elemento com características de **display** iniciais pode ser re-configurado

```
a{  
    display : block;  
} /*inline agora como block */
```

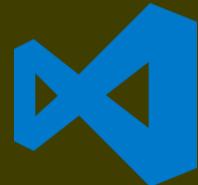
```
h1{  
    display : inline;  
} /* block agora como inline */
```

```
div{  
    display : inline-block;  
} /* block agora como inline-block */
```

A situação mais comumente encontrada é a re-configuração de um elemento de display block para display inline-block

Importante: estude os exemplos mostrados pelo professor no codepen.io, faça alterações e observe o resultado da renderização!

Conflitos entre regras CSS



< a :link>

Selector Specificity: (0, 1, 1)

a:link{ font-size: 26px }

- No caso de regras conflitantes, o que irá contar é o **seletor**:

```
<p id="promocao" class="vermelho">  
    Camisas vermelhas terão 30% de desconto hoje!  
</p>
```

Como o browser irá resolver o conflito?

- O browser irá primeiro selecionar todas as regras que se aplicam (*matches*) ao parágrafo
- Calcular o peso da regra (sua especificidade)
- Re-ordenar as regras observando o peso
- Se houver pesos iguais, valerá a última regra (a ordem na declaração de cima para baixo)

- Como calcular?

Basta adicionar o número 1 para cada item encontrado na regra CSS declarada

0	0	0
id	class	elemento

.vermelho	0	1	0
p.vermelho	0	1	1
h3	0	0	1

```
.vermelho{  
    font-size : 18px;  
}  
#promocao{  
    font-size : 26px;  
}  
p{  
    font-size : 32px;  
}
```

Acima, há regras conflitantes!
Qual será o tamanho da fonte do elemento na tela, em px?

Exemplos

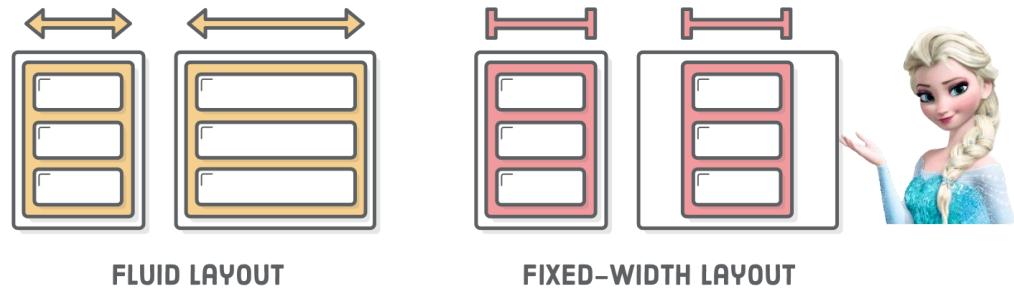
div	h1	0	0	2
#menu	p	1	0	1
#promocao		1	0	0



Layout e posicionamento

- Há diversas maneiras de conceber um layout para uma página
 - Pode-se usar um **layout “líquido”**: o conteúdo principal da página poderá “crescer” ou “diminuir”, conforme a janela do navegador é redimensionada
 - Este layout possui algumas limitações (vide práticas anteriores)
 - Pode-se usar um **layout “frozen”**: o tamanho do conteúdo principal da página não será alterado, quando a janela do navegador for redimensionada
 - Usa não só a propriedade **width**, mas **margin** também

Não há um layout perfeito: a escolha recai no que o site deve permitir o usuário fazer e o webdesigner saber ou não implementá-lo. O layout frozen também possui limitações, mas é bastante utilizado



- Nos próximos slides, você irá praticar diferentes layouts usando técnicas modernas

Float, Flexbox & CSS Grid

- Layout é a maneira com que conteúdos como textos, imagens e outros elementos HTML são posicionados (“arranjados”) na página
 - A ideia é usar layouts para criar uma estrutura visual, em vez de deixar o browser usar o fluxo padrão para posicionar e exibir os elementos
- Nos dias atuais, CSS oferece três técnicas para a criação de layouts:

Float, Flexbox e CSS Grid

- **Float** layouts vêm sendo substituído nos últimos tempos por **FlexBox** e **CSS Grid**
 - Considerado obsoleto, mas bastante utilizado no layout de muitas páginas na Web
- **Flexbox** é uma forma moderna de se criar layouts, em **particular no arranjo de componentes “internos” às grandes seções lógicas da página (>granularidade)**

¹ <https://developer.mozilla.org/en-US/docs/Web/CSS/float>

² https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout/Basic_concepts_of_flexbox

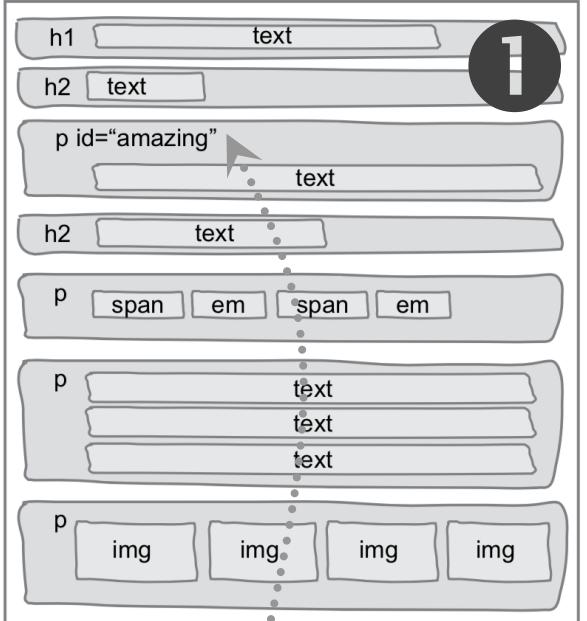
³ https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout

Float, Flexbox & CSS Grid (cont.)

- **Float, Flexbox e CSS Grid (cont.)**
 - Flexbox não utiliza a float property, não obriga o webdeveloper a posicionar as seções “flutuantes” abaixo de outras dentro do arquivo HTML (como Float faz)
 - CSS Grid é outra forma moderna de arranjar elementos HTML na página em grades
 - Aqui, a ideia é organizar a estrutura visual da página em 2-dimensões (row, column)
 - Bom para criar layouts e posicionar as grandes seções lógicas da página (< granularidade)
 - Entretanto, há sites que usam Flexbox nas grandes áreas lógicas da página
 - Não existe um consenso sobre “o melhor layout” ou a “melhor abordagem”
- Além das três abordagens acima, outra forma de organizar elementos na página é utilizando o absolute positioning
 - Será mostrado mais adiante nos próximos slides...

Float Layout

- Lembre que browsers posicionam elementos usando **fluxo padrão (normal flow)**
 - A partir do topo, até a parte inferior da área visível da página, com uma quebra de linha antes e depois de cada elemento de bloco (não flutuante) encontrado
 - Por default, cada elemento de bloco irá ocupar toda a largura da área visível do navegador
 - Elementos “inline” (ou em linha), por sua vez, fluem a partir da parte superior esquerda do elemento de bloco até encontrar sua parte inferior direita (default)
- Elementos são capazes de **flutuar** (com **float**) acima do fluxo padrão
 - A idéia é “descolar” um elemento da página podendo movê-lo para esquerda ou à direita
 - Lembre-se: se, por um lado, eles não são mais “enxergados” pelos elementos de bloco, por outro, elementos inline irão considerar e respeitar os seus limites (contornando-os)
 - Para flutuar, um elemento deve ter um **id** e um **width** <https://codepen.io/C-sar-Rocha/pen/zYydvVx>
 - A desvantagem de elementos flutuantes é que, dentro do HTML, devem estar logo após o elemento a partir do qual devem “flutuar-abaixo-de” (no HTML, isso é estranho no caso de sidebar e header)

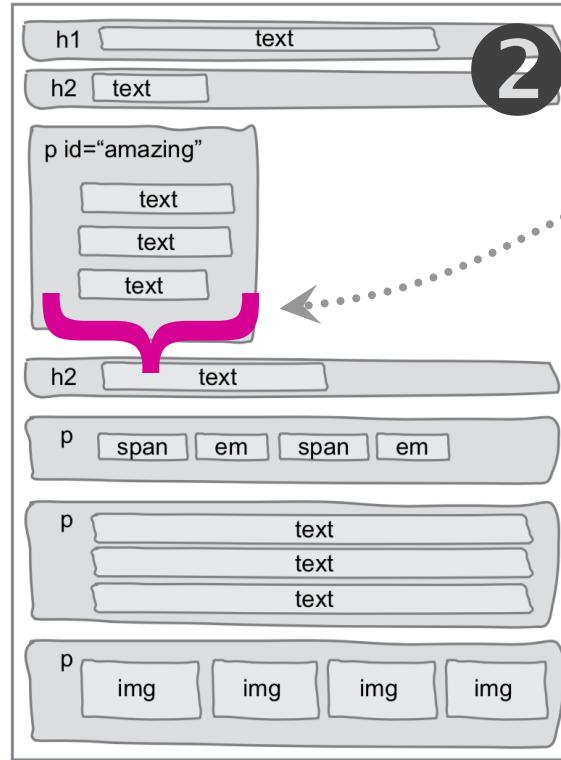


`<p id="amazing"> ... </p>`
arquivo.html

Pois o elemento que irá flutuar deverá ter um **id** & **width**

`#amazing {
 width: 200px;
 float: right;
}`

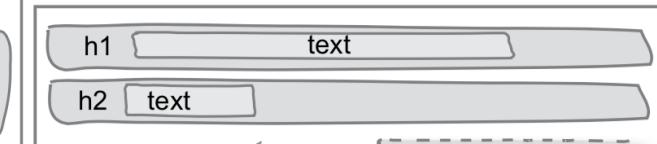
float¹ pode ser **none**, **left** ou **right**.



```
#amazing {  
    width: 200px;  
}
```

arquivo.css

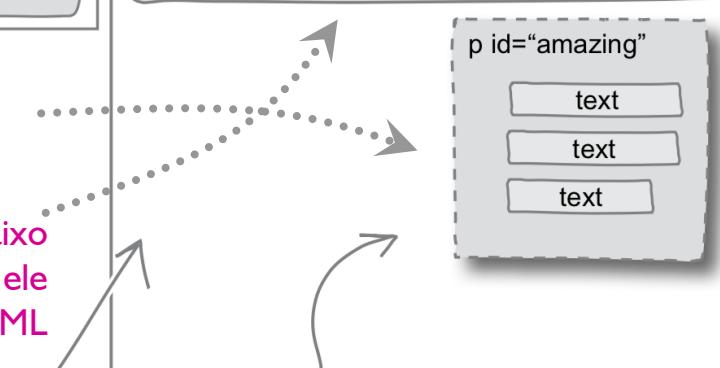
Texto será ajustado por conta de **width** ter sido “encurtado”. Note que `<p>` ainda pertence ao fluxo normal (linebreak antes e depois)



Note que `<p>` agora não pertence ao fluxo normal

`<p>` irá flutuar abaixo de `<h2>` pois é onde ele está dentro no HTML

(1) First, the browser flows the elements on the page as usual, starting at the top of the file and moving toward the bottom.



(2) When the browser encounters the floated element, it places it all the way to the right. It also removes the paragraph from the flow, like it's floating on the page.

4

¹ <https://developer.mozilla.org/en-US/docs/Web/CSS/float>

* Exemplo obtido do livro Head First HTML CSS, 2nd Edition



... importante observar que os inline elements irão respeitar os limites do parágrafo “flutuante”....

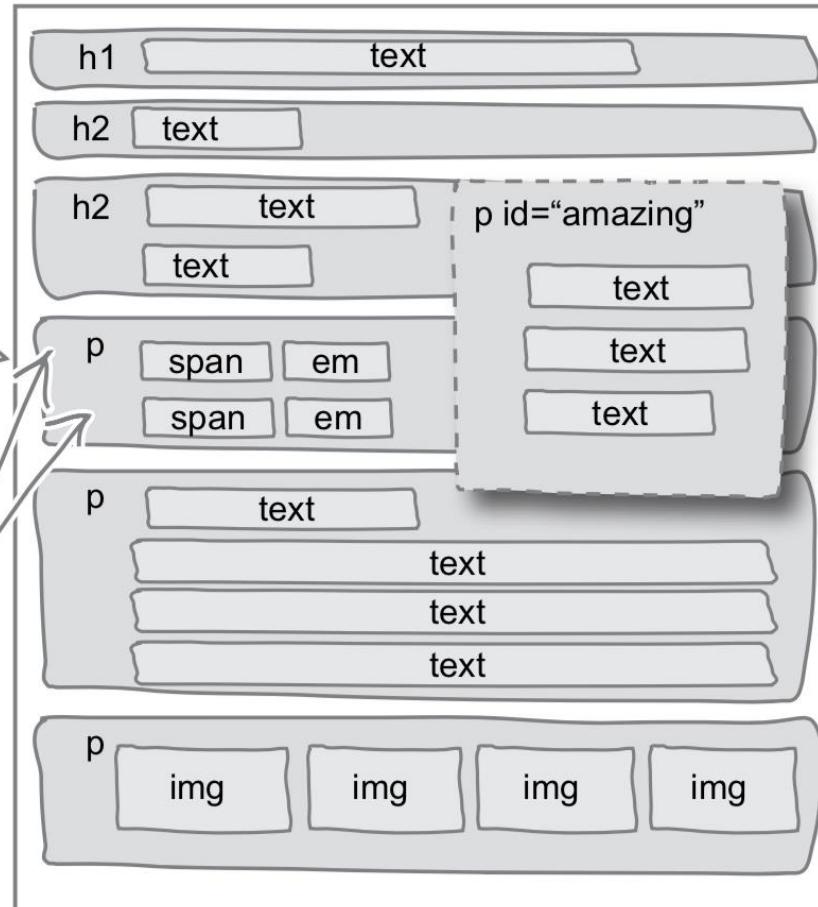
... e que os demais block elements irão ser “deslocados para cima” e posicionados “abaixo” do floated parágrafo...

... os primeiros <h1> e <h2> irão ser exibidos normalmente, no fluxo normal, até o browser encontrar o <p> flutuante...

(3) Because the floated paragraph has been removed from the normal flow, the block elements are filled in, like the paragraph isn't even there.

(4) But when the inline elements are positioned, they respect the boundaries of the floated element. So they are flowed around it.

... mas, se <p> tivesse sido movido para o primeiro elemento na página html, ele iria “subir” ainda mais na visualização. E os textos dos primeiros <h1> e <h2> já teriam de observar/respeitar o floated <p>, com texto “ao redor” dele ...



... ou seja, as caixas de elementos block ainda irão passar por baixo do floated <p>, mas o conteúdo irá respeitar os limites do <p> flutuante, quando forem exibidos...

Notice that the block elements are positioned under the floated element. That's because the floated element is no longer part of the normal flow.

However, when the inline elements are flowed within the block elements, they flow around the borders of the floating element.

... no HTML, deve-se de colocar o element floated imediatamente abaixo daquele que você quer que o floated seja exibido “abaixo-de” no fluxo; ainda que no corpo <body>

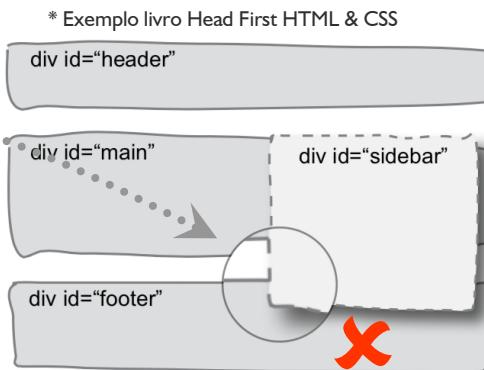


Propriedade clear (evitar overlapping)

- No layout sidebar (à direita) é bastante comum ter um overlapping indesejado entre o **float** elemento e os outros elementos que ficaram no normal flow

Ao aumentar a largura da janela do browser o **#footer** irá “subir” e fazer overlapping com *floated sidebar*

```
#sidebar {  
    width: 200px;  
    float: right;  
}  
arquivo.css
```



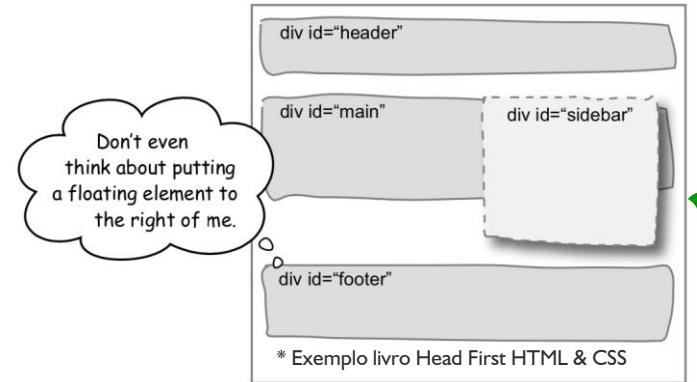
Isso acontece porque **#sidebar** foi removida do normal flow. **#main** e **#footer** estão no normal flow e são exibidos embaixo, ignorando a **#sidebar** (exceto, claro, texto e inline elements que irão respeitar a **#sidebar** contornando-a, ao redor dela)

- Porém, aplique a propriedade clear do CSS no elemento, para evitar overlapping¹
 - Essa propriedade define se um elemento deve ser movido para baixo (cleared) de outro floated elemento que o precede.

```
#footer {  
    ...  
    clear: right;  
}  
arquivo.css
```

clear¹ pode ser none, left , right e both

Se houver algum *floated* elemento à direita de **#footer**, **#footer** será movido um pouco para baixo até não “encostar” mais em algo; **#footer** estará sempre abaixo de **#sidebar**



¹ <https://developer.mozilla.org/en-US/docs/Web/CSS/clear>

Recapitulando...

- Os elementos são localizados na página usando fluxo normal

- **float¹** elements serão removidos do normal flow e colocados à esq. ou à dir.

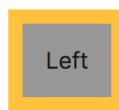
As much mud
in the streets
as if the waters
had but newly
retired from
the face of the earth, and
it would not be wonderful
to meet a Megalosaurus,
forty feet long or so,
waddling like an
elephantine lizard up
Holborn Hill.



- Devem ter as propriedades **id** e **width** configuradas nos arquivos HTML e CSS
- “Flutuam” acima dos elementos de bloco e não interferem em seu fluxo: a exceção é quando se usa a propriedade **clear²** em um block element, o que irá causar o movimento para baixo desse block elem. até não encostar mais no float elem. (seja pela esq., dir., ou ambos os lados)
- Porém, texto & inline sabem onde **float** elements estão, e se acomodam “ao redor” dele
- Deve-se re-ordenar o HTML, para que a #sidebar seja exibida abaixo do elemento desejado
 - » Embaixo do header, e.g.

Inline & texto respeitam o **float** element
e o **flow** é contornar “ao redor”

Ao lado, três cenários de uso da propriedade **clear**: ela irá causar movimentação do elemento de bloco <p> situado embaixo (normal flow) do float element (caixa laranja)



As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.



As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.



As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

clear: left; **clear: right;**

clear: both;

¹ <https://developer.mozilla.org/en-US/docs/Web/CSS/float>

² <https://developer.mozilla.org/en-US/docs/Web/CSS/clear>



Um exemplo breve...

Laboratório I2: layout “barra lateral” na página
fictícia do site petstore.com



Absolute positioning layout

```
#sidebar {  
    position: absolute;  
    width: 280px;  
    top: 100px;  
    right: 200px;  
}  
arquivo.css
```

Note top & right properties

Because sidebar is now absolutely positioned, it is removed from the flow and positioned according to any top, left, right, or bottom properties that are specified.

Para voltar ao normal flow, basta configurar position: relative

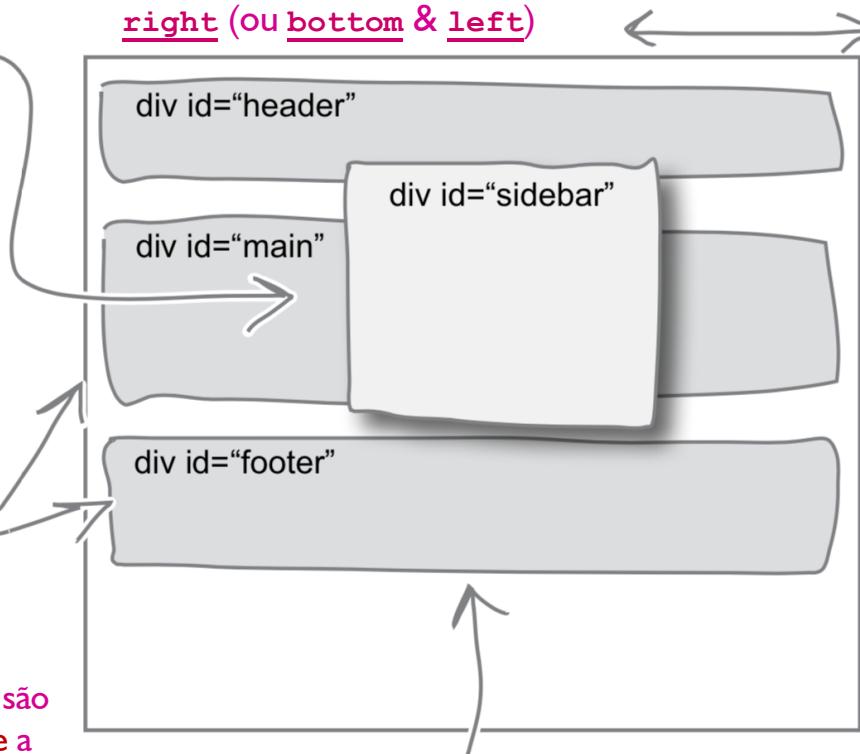
Because the sidebar is out of the flow, the other elements don't even know it is there, and they ignore it totally.

Diferente de float, elementos do normal flow são exibidos embaixo, mas ignorando totalmente a #sidebar (texto e inline elements NÃO irão respeitar a #sidebar e NÃO irão contorná-la -- ao redor dela)

#sidebar será retirado do fluxo normal, com absolute

O browser irá posicionar #sidebar (onde você quiser!) usando top e right (ou bottom & left)

The sidebar is positioned 200 pixels from the right of the page.



And the sidebar is positioned 100 pixels from the top of the page.

#sidebar pode "cobrir" ficar acima do conteúdo de elementos abaixo

Local final de #sidebar é definido por top, left, bottom ou right

Elements that are in the flow don't even wrap their inline content around an absolutely positioned element. They are totally oblivious to it being on the page.



* Exemplo obtido do livro Head First HTML CSS, 2nd Edition

[1 https://developer.mozilla.org/en-US/docs/Web/CSS/position](https://developer.mozilla.org/en-US/docs/Web/CSS/position)

Absolute positioning layout (cont.)

- A **position** permite configurar como um elemento é posicionado dentro da página HTML. Os valores posicionais **top**, **right**, **bottom** e **left** ajustam a localização final do elemento posicionado
 - **position: static**¹, **relative**, **absolute**, **fixed** e **sticky**
 - **static**: elemento pertence ao normal flow, mas propriedades posicionais não têm efeito;
 - **relative**: elemento pertence ao normal flow (como static), mas valores posicionais “empurram” o elemento da sua posição original na direção do valor da propriedade posicional antes de exibi-lo;
 - **absolute**: elemento é removido do normal flow. Outros elementos (até texto, etc.) o irão ignorar, como se o absolute element simplesmente não estivesse ali; Valores posicionais têm efeito;
 - **sticky**: semelhante a relative, mas sua exibição “se move” com scrolling da tela na viewport;
 - **fixed**: elemento é removido do normal flow (como absolute), com a diferença que fixed elements não se movem com scrolling da viewport (área visível da página)

Para um element voltar ao normal flow, use **position: relative**; Isso significa que elementos serão arranjados na ordem como aparecem dentro do arquivo HTML



¹ static é o default para a propriedade position

Absolute positioning layout (cont.)

- No absolute positioning, **top**, **right**, **bottom** e **left** ajustam a localização final do elemento posicionado a partir da viewport*
 - Ou seja, a partir da área visível (corrente) da página HTML
- Porém, para que o elemento absolute seja posicionado a partir do seu parent container, configure a position do parent com relative

```
body {  
    ...  
    position: relative;  
}
```

Propriedades posicionais
top, **left**, **right** e **bottom**
agora serão relativas ao
<body> e não mais ao
viewport

```
#sidebar {  
    position: absolute;  
    width: 280px;  
    top: 100px;  
    left: 200px;  
}  
/* top,right,etc. relativo ao body */
```

absolute irá remover a
#sidebar do normal flow

div#sidebar pode estar em
qualquer lugar do código html

```
<!doctype html>  
  
<html lang="pt">  
    <head> ... </head>  
    <body>  
        <div id="sidebar">  
            ...  
        </div>  
        <p>...</p>  
    </body>  
</html>
```

arquivo.html



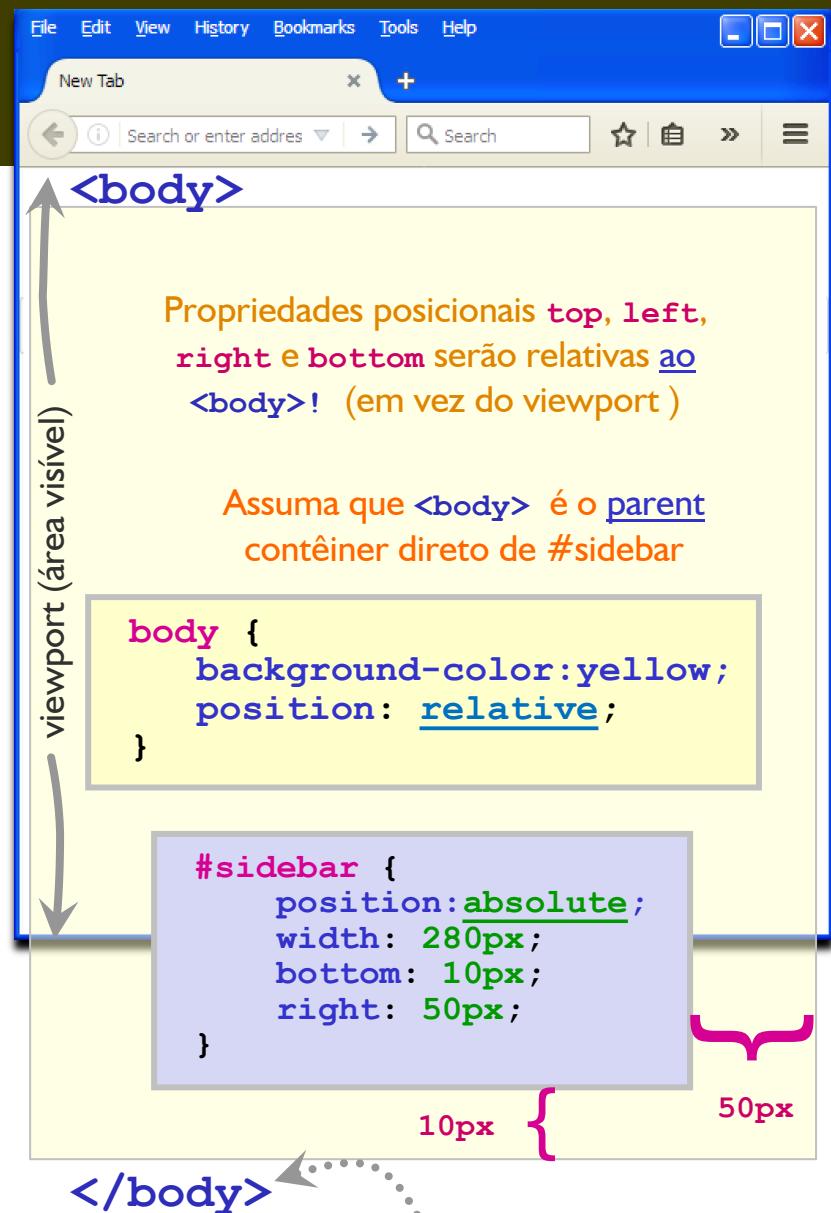
Absolute (graficamente)...

The diagram illustrates the absolute positioning of the #sidebar element relative to the viewport. A large yellow box represents the viewport (área visível). Inside it, a smaller purple box represents the body element. The #sidebar element is positioned absolutely within the body area. Two arrows point upwards from the bottom of the sidebar towards the top of the body box, indicating the relative position of the sidebar relative to the body's top edge.

```
body {  
    background-color:white;  
    position: relative;  
}  
  
#sidebar {  
    position: absolute;  
    width: 280px;  
    bottom: 10px;  
    right: 50px;  
}
```

viewport (área visível)

#sidebar será posicionada no final da viewport, ainda que o final da página html ainda não esteja sendo exibido



#sidebar será posicionada no final do corpo da página, 88 ainda que a viewport não esteja exibindo o final da página html



Recapitulando...

- Comparando as diferentes estratégias de layout vistas até aqui:

Normal flow

- Posicionamento padrão;
- Elementos são posicionados pelo browser, a partir do topo, até a parte inferior da página
- Quebras de linha entre block elements. Estes ocuparão toda a largura disponível
- Inline elements fluem da esquerda para à direita
- A ordem de posicionamento é a ordem dentro do HTML

Float

- Elementos são retirados do normal flow pelo browser
- Usa-se float: left ou right
- Flutuam acima de block elements que ficaram no normal flow (exceto se usar clear)
- Inline elements & texto fluem ao redor de float elements;
- Deve-se reordenar o HTML (para colocar o float element abaixo daquele que se deseja flutuar-abaixo-de na página)

Absolute

- Elementos são retirados do normal flow pelo browser
- Usa-se position: absolute;
- Flutuam acima de quaisquer elementos que ficaram no normal flow (embaixo)
- Os elementos não sabem que absolute element está acima;
- Absolute element pode estar em qualquer lugar dentro do HTML
- Usa-se top, left, right e bottom para ajustar a posição final;
- A posição é feita pelo viewport ou pelo parent container (relative);

Cuidado: child-elements que são retirados do flow podem colapsar¹ o parent element. Estude o exemplo no CodePen:

¹ <https://codepen.io/C-sar-Rocha/pen/ExGvdVv>



Um exemplo breve...

Laboratório I3: layout complexo “colorido” usando
absolute positioning



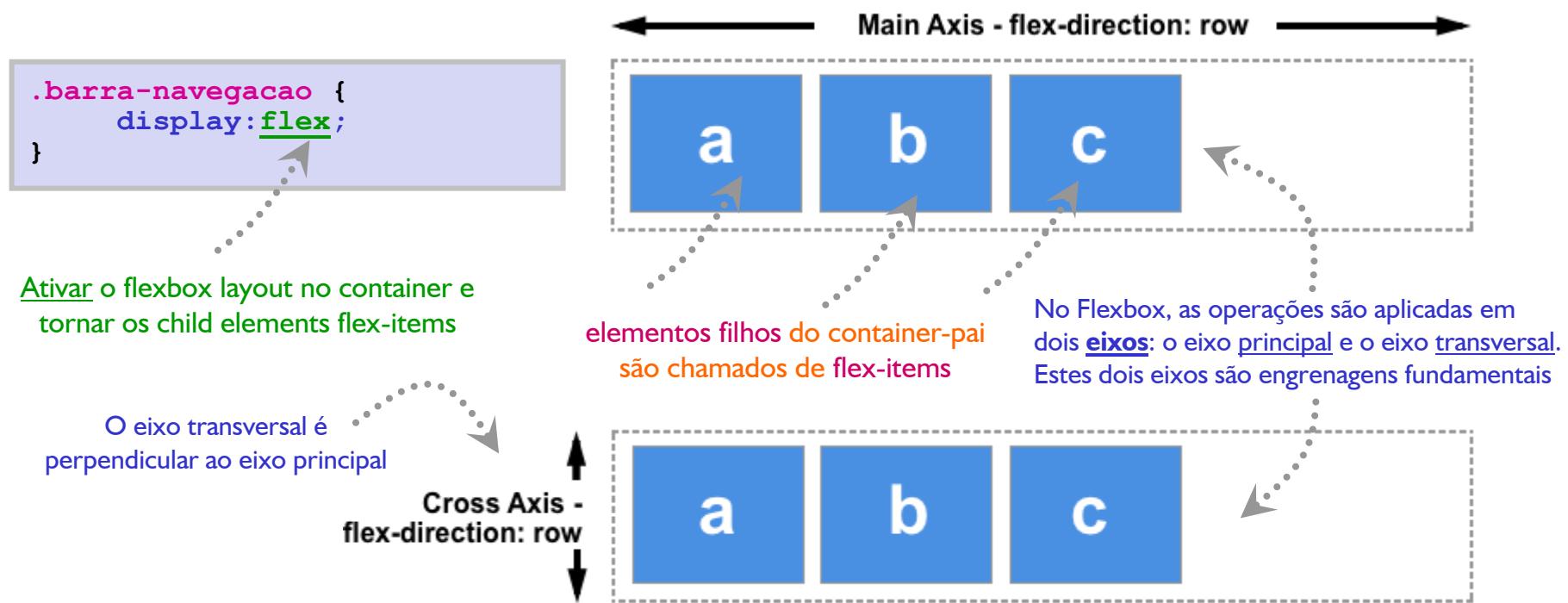
Flexbox layout

- O “módulo CSS de layout de caixa flexível” (Flexbox)
 - Considerado uma das formas modernas de se criar layouts usando CSS
 - A ideia é que o browser (e não você) organize, calcule e divida o espaço dentro de um contêiner-pai entre seus elementos filhos de forma automática
 - Unidimensional: controlar filhos em uma dimensão por vez: linhas ou colunas
 - Flexbox permite alinhar verticalmente/horizontalmente os itens filhos dentro de um contêiner-pai facilmente (sem especificar coordenadas em pixels)
 - Ajuda a reduzir a complexidade de se criar layouts mais profissionais
 - Colunas com width iguais, alinhamento horizontal ou vertical entre elementos, eram problemas comuns e difíceis de implementar com outras técnicas (e.g., Float)

¹ https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout/Basic_concepts_of_flexbox

Flexbox layout (cont.)

- O elemento controlado pelo Flexbox é chamado flexbox container



O eixo principal é definido através da propriedade **flex-direction**. Ao ativar a propriedade **display: flex**, os child elements irão ser **flex-direction: row**

¹ Figuras obtidas da documentação oficial Flexbox

¹ https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout/Basic_concepts_of_flexbox

Flexbox layout (cont.)

Eixo principal:

A propriedade **flex-direction** define a direção do eixo principal

```
.barra-navegacao {  
    display:flex;  
    flex-direction:row;  
}
```

Pode ser row, row-reverse , column e column-reverse

```
.barra-navegacao {  
    display:flex;  
    flex-direction: column;  
}
```

Note que o eixo principal agora se moverá ao longo da coluna na direção block

A direção em que os child elements serão organizados será em termos do eixo principal

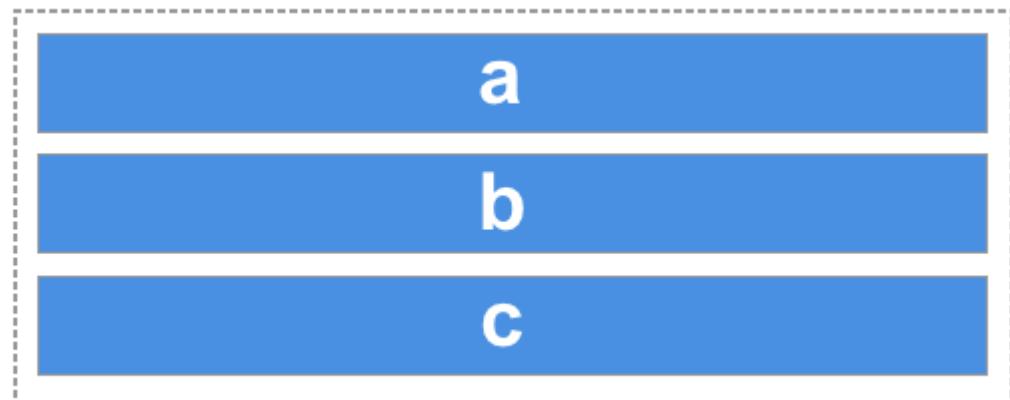
Por default, child elements irão ocupar apenas o espaço necessário para exibir o conteúdo. Contudo, flex-items ocuparão a altura, isto é, serão “tão altos” quanto o mais alto elemento (seja o flexbox container parent ou outro flex-item irmão)

Note que o eixo principal se moverá ao longo da linha na direção inline
Os child elements irão ser organizados um ao lado do outro (em linha)



¹ Figuras obtidas da documentação oficial Flexbox

Os child elements irão ser organizados um acima do outro (em coluna)



¹ Figuras obtidas da documentação oficial Flexbox

¹ https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout/Basic_concepts_of_flexbox



Flexbox layout (cont.)

```
.barra-navegacao {  
    display: flex;  
    flex-direction: row;  
}
```

O eixo transversal é sempre perpendicular ao eixo principal

Cross Axis -
flex-direction: row

Eixo transversal:

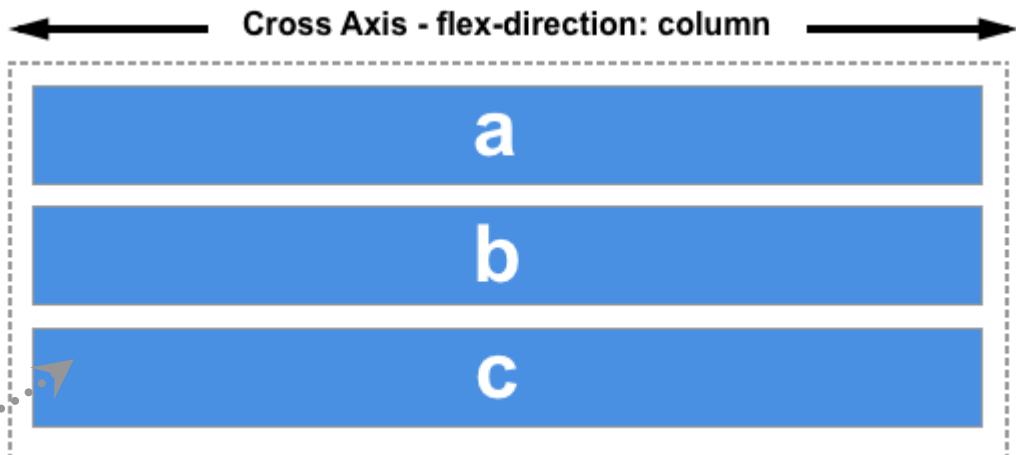
Se a propriedade flex-direction estiver definida nas linhas, e.g., row ou row-reverse, o eixo transversal estará na direção das colunas;



¹ Figuras obtidas da documentação oficial Flexbox

```
.barra-navegacao {  
    display: flex;  
    flex-direction: column;  
}
```

Se a propriedade flex-direction estiver definida nas colunas, e.g., column ou column-reverse, o eixo transversal estará na direção das linhas;



¹ Figuras obtidas da documentação oficial Flexbox



Listagem de exemplos de uso com as principais propriedades do Flexbox



```
/* habilita o uso do flexbox em um container */  
display: flex
```

<https://codepen.io/C-sar-Rocha/pen/OJrEWNQ>

```
/* define a direção do eixo principal (main axis) */  
flex-direction: row | row-reverse | column | column-reverse
```

<https://codepen.io/C-sar-Rocha/pen/PoXdZLL>

```
/* define o alinhamento dos itens eixo principal (main-axis); horizontalmente, por default */  
justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly
```

<https://codepen.io/C-sar-Rocha/pen/mdaGrmL>

```
/* define o alinhamento dos itens eixo transversal (cross-axis); verticalmente, por default */  
align-items: stretch | flex-start | flex-end | center
```

<https://codepen.io/C-sar-Rocha/pen/GRPXjxX>

```
/* deslocamento de itens para outra nova linha (em cima ou abaixo) quando não houver espaço */  
flex-wrap: nowrap | wrap | wrap-reverse
```

<https://codepen.io/C-sar-Rocha/pen/BavOWyV>

```
/* só é aplicável quando se tem múltiplas linhas (e flex-wrap:wrap)  
modifica o comportamento de flex-wrap: similar à propriedade align-items; mas, em vez de alinhar  
itens individuais, esta propriedade irá alinhar/puxar as linhas (wrapped-lines) */  
align-content: stretch | flex-start | flex-end | center | space-between | space-around
```

<https://codepen.io/C-sar-Rocha/pen/gOZdgBx>

```
/* criar manualmente um gap (espaço) entre itens, sem usar margin */  
gap: 0 | <length>
```

<https://codepen.io/C-sar-Rocha/pen/eYbLgVX>

As propriedades acima são aplicadas em **flex-containers**

Atenção: **flex-start** leva em consideração a presença do "-reverse" em flex-direction, enquanto que só o valor **start** não o faz



Listagem de exemplos de uso com as principais propriedades do Flexbox



```
/* permite fazer override de align-items em um flex-item individual*/  
align-self: stretch | flex-start | flex-end | center
```

<https://codepen.io/C-sar-Rocha/pen/gOZBGyO>

```
/* permite mudar a ordem de flex-items dentro do flex-container sem mudar no HTML */  
order: <int>
```

<https://codepen.io/C-sar-Rocha/pen/dywglAV>

```
/* recomendação quanto ao tamanho width dos itens dentro do flex-container */  
flex-basis: auto | min-content | max-content | fit-content
```

<https://codepen.io/C-sar-Rocha/pen/mdaGrmL>

```
/* define se os flex-items podem “encolher” para não haver overflow no flex-container */  
flex-shrink: 1 | 0
```

<https://codepen.io/C-sar-Rocha/pen/XWoOYGd>

```
/* define se flex-items podem “crescer” e preencher o espaço restante do flex-container “fill” */  
flex-grow: 0 | 1 <integer>
```

<https://codepen.io/C-sar-Rocha/pen/LYMqByd>

<https://codepen.io/C-sar-Rocha/pen/KKbJBrN>

```
/* atalho para flex-grow, -shrink, -basis */  
flex: 0 1 auto | <int><int><length>
```



As propriedades acima são aplicadas em **flex-items**

Um exemplo breve...

Laboratório I4: FlexBox “challenge”



CSS Grid layout

- **CSS Grid** é uma forma alternativa para se criar layouts no CSS
 - Também é considerada uma técnica moderna e não substitui o Flexbox
 - A ideia é que o browser organize, automaticamente, todo o espaço dentro de um contêiner-pai na forma de uma grade (ou grid) com várias células
 - Bi-dimensional: controle feito em duas dimensões: linhas , colunas e gaps (espaços)
 - Da mesma forma que o Flexbox, CSS Grid permite alinhar os itens filhos dentro das células facilmente (sem especificar coordenadas em pixels)
 - Pode-se criar grids fixos ou expansíveis rapidamente -- sem ter de calcular valores para as propriedades width e height para dividir espaços em linhas e colunas
 - CSS Grid permite fazer merge (span) do conteúdo em múltiplas células

¹ https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Grids

CSS Grid layout (cont.)

- O elemento controlado pelo CSS Grid é chamado grid container

Ao ativar a propriedade `display:grid` um grid é criado com uma única coluna para os child-elements. Logo, de início, não se perceberá nenhuma diferença.

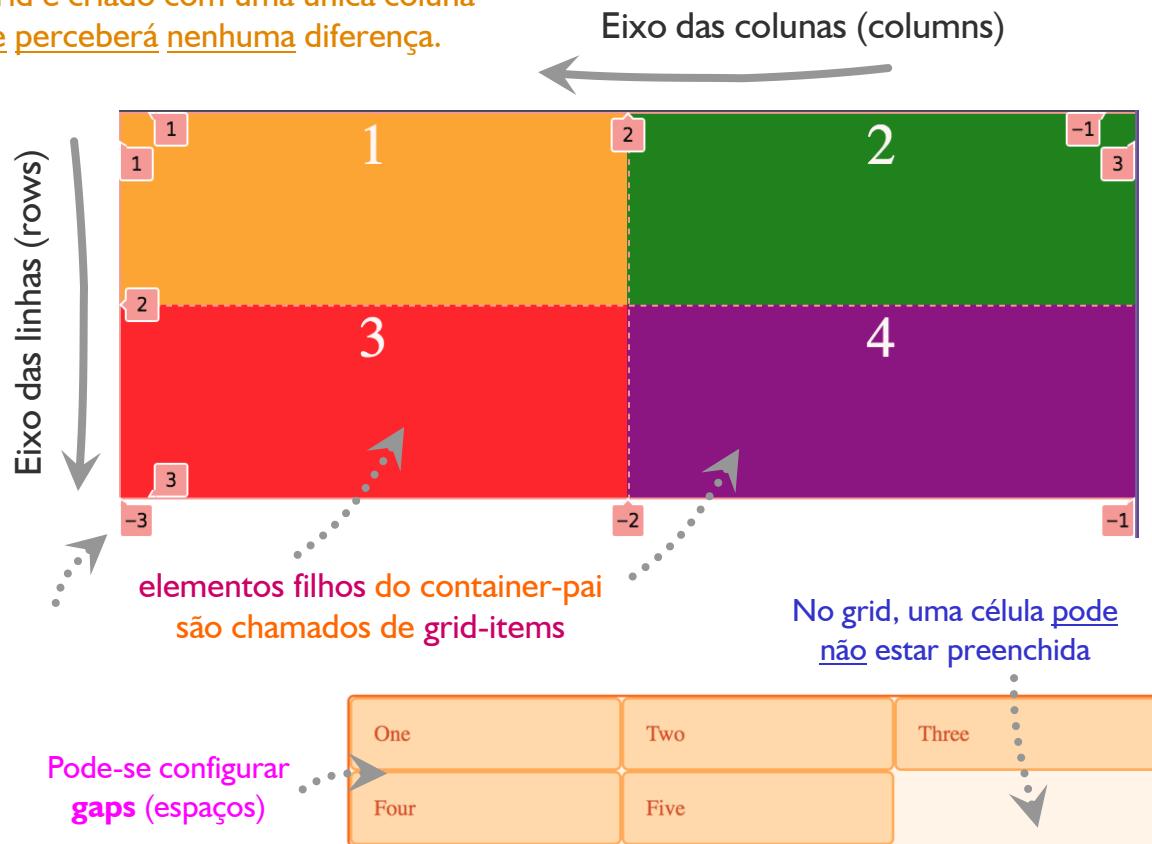
```
div#fotos {  
    display:grid;  
}
```

Ativar o CSS Grid layout no container e tornar os child-elements grid-items

Propriedades `grid-template-columns` e `grid-template-rows` devem ser aplicadas para se montar o grid desejado

Os números são coordenadas – úteis para mover ou fazer spanning de células

Diferente do Flexbox, onde se podia mudar a disposição dos eixos principal e transversal nas operações aplicadas, no CSS Grid os eixos não mudam (eixos linha e coluna são sempre fixos)



¹ Figura obtida da documentação oficial CSS Grid



Listagem de exemplos de uso com as principais propriedades do CSS Grid



```
/* habilita o uso do grid em um container */  
display: grid
```

<https://codepen.io/C-sar-Rocha/pen/gOqbZxg>

```
/* define as linhas e colunas do grid; unidades comuns: px (fixo), fr (extensível) e auto */  
grid-template-rows: <track size>  
grid-template-columns: <track size>
```

<https://codepen.io/C-sar-Rocha/pen/PoXdZLL>
<https://codepen.io/C-sar-Rocha/pen/jjxowao>
<https://codepen.io/C-sar-Rocha/pen/ExraGqa>

```
/* criar manualmente um gap (espaço) entre as tracks de linhas e colunas, sem usar margin */  
row-gap: 0 | <length>  
column-gap: 0 | <length>
```

<https://codepen.io/C-sar-Rocha/pen/dyaYXpd>

```
/* alinhar os itens dentro das células linhas/colunas (horizontalmente/verticalmente) */  
justify-items: stretch | start | center | end  
align-items : stretch | start | center | end
```

<https://codepen.io/C-sar-Rocha/pen/mdvyZdQ>

```
/* alinhar o grid que está dentro do grid-container; só faz sentido se container é maior que o grid */  
justify-content: start | center | end  
align-content : start | center | end
```

<https://codepen.io/C-sar-Rocha/pen/ZEwYNPW>

As propriedades acima são aplicadas em **grid-containers**

Atenção: ao configurar só colunas, caso haja poucas para acomodar uma grande quantidade de conteúdos, linhas novas podem ser criadas automaticamente (linhas implícitas) apenas para acomodar todo o conteúdo



Listagem de exemplos de uso com as principais propriedades do CSS Grid



```
/* para posicionar (mover) um grid-item para uma célula específica, usando as coordenadas das tracks  
a keyword <span> pode ser usada para fazer merge de um grid-item em múltiplas células */  
grid-column: <start> / <end>  
grid-row: <start> / <end>
```

<https://codepen.io/C-sar-Rocha/pen/BaMyepP>

```
/* faz override de eventuais valores existentes de propriedades <justify-items/align-items>  
apenas em um grid-item específico */  
justify-self: stretch | start | center | end  
align-self : stretch | start | center | end
```

<https://codepen.io/C-sar-Rocha/pen/oNmgrxy>



As propriedades acima são aplicadas em **grid-items**

Um exemplo breve...

Laboratório 15: FlexBox “challenge” agora sendo configurado com CSS Grid



Para um melhor aproveitamento...

- Ao codificar exemplos mostrados em sala, procure verificar pontos de dúvidas com o professor.
- Não estude apenas por slides (são apenas tópicos)!
- Mantenha em dia todas as questões das **listas de exercícios e práticas de laboratório**.
- Não se acanhe! Procure-me (ou monitor da disciplina, quando houver algum) e questione conceitos, listas, etc.
- Não deixe para começar a estudar em cima da hora.



Bibliografia

- DAMASCENCO, Aniele. Webdesign Teoria e Prática. Editora Visual Books, 2003.
- MEMÓRIA, Filipe. Design para Internet. Editora Campus, 2005.
- NIEDERST, Jennifer. Aprenda Web Design. Editora Ciência Moderna, 2002.
- SCHMIDT, Paulo. Fundamentos de Auditoria de Sistemas. Editora Atlas, 2006.
- Artigos e materiais na Web.

