

Layouts

Tipos

Abordagens

Layout

- **Tabelas (Deve ser evitado!)**
 - Forma “tradicional” de estruturar um layout. Origina código confuso, problemas de redimensionamento. Evitar!!
- *float /clear*
 - Implementação simples. Ligado ao fluxo do HTML o que reduz a sua flexibilidade
- **CSS FlexBox**
 - Baseia-se num conjunto de propriedades que permitem a disposição flexível dos elementos ao longo de um eixo.
 - Só suportado a partir da versão 11 do IE (suportado a partir de 2015)
- **CSS Grid Layout**
 - Baseia-se na criação de um grid container para disposição dos elementos por linha/coluna
 - Suportado pelos browsers (sem vendor prefix) a partir de 2017
- **Frameworks CSS**
 - Forma rápida e poderosa de criação de um layout (exemplo:Bootstrap)
 - Aprendizagem de um novo framework

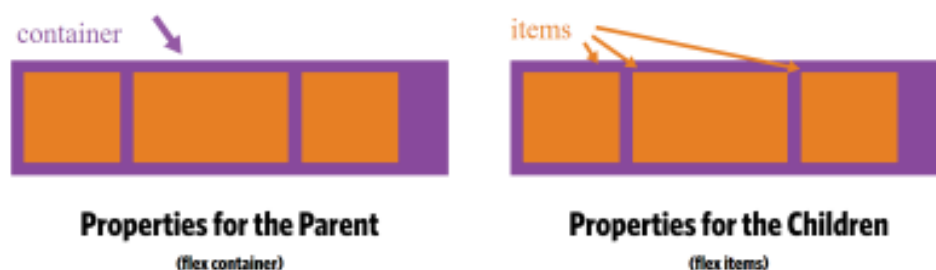
CSS Flexbox (Layouts)

CSS Flexbox

- *flexbox container*
 - permite agilizar a definição de layouts

According to the specification, the Flexbox model provides for an efficient way to **layout, align, and distribute space among elements within your document** — even when the viewport and the size of your elements is dynamic or unknown.

<https://medium.freecodecamp.org/understanding-flexbox-everything-you-need-to-know-b4013d4dc9af>



- `{display:flex}`
 - cria um *container* flexível com um conjunto de propriedades para o *container* e para os respetivos *items*

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

- Pode ser aplicado a diversos elementos

- *ul* : flex container
- *li* : flex item (filhos do flex container)

```
<ul>  
  <li>1</li>  
  <li>2</li>  
  <li>3</li>  
</ul>
```

```
ul{display:flex;}  
  
li {  width: 100px;  
      height: 100px;  
      background-color:lightblue;  
      margin:8px;  
  
      list-style-type:none;  
      text-align:center;  
      color:white}
```



CSS Flexbox (Layouts)

flex container

flex item

■ Propriedades do **flex container**:

■ **flex-direction**

- Estabelece a direção do eixo principal ao longo do qual os elementos são dispostos

■ **flex-wrap**

- Define se o *container* adapta a sua dimensão ou se cria múltiplas linhas para conter os *flex items*

■ **flex-flow**

- propriedade agregada de *flex-direction* e *flex-wrap* (ex: `flex-flow: row wrap;`)

■ **align-items**

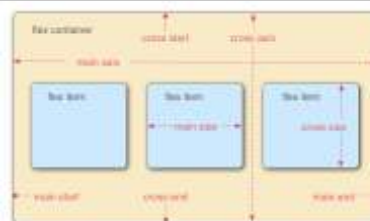
- define o alinhamento dos *items* ao longo do *cross axis*

■ **align-content**

- define o alinhamento dos *items* ao longo do *cross axis* quando existe mais do que uma linha

■ **justify-content**

- define o alinhamento dos *items* ao longo do *main axis*



■ Propriedades dos **flex container**

- **flex-direction**: row | row-reverse | column | column-reverse



- **flex-wrap**: nowrap | wrap | wrap-reverse

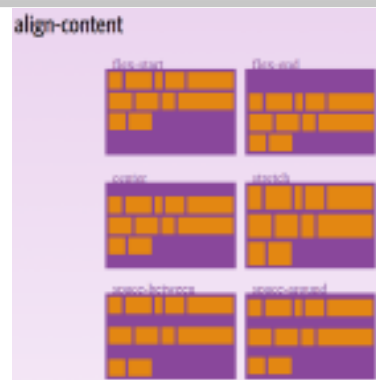
- O comportamento por defeito é nowrap, o flex container adapta-se aumentando de tamanho para conter todos os flex items
- O valor wrap cria múltiplas linhas



- **align-items**: flex-start | flex-end | center | space-between | space-around | stretch;



- **align-content:** flex-start | flex-end | center | space-between | space-around | stretch;
 - Não se aplica quando existe apenas uma linha de *items*



- **justify-content:** flex-start | flex-end | center | space-between | space-around | space-evenly;



<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

Exemplo:

```
<div id="container">
  <div id="d1">Item1 Item1 Item1 Item1 Item1 Item1 Item1 </div>
  <div id="d2">Item2 Item2 Item2 Item2 Item2 Item2 Item2 </div>
  <div id="d3">Item3 Item3 Item3 Item3 Item3 Item3 Item3 </div>
</div>
```

```
#container{
  height:400px;
  width:400px;
  display:flex;
  flex-direction: row;
  align-items:flex-start;}
```

Item1	Item1	Item1	Item2	Item2	Item2	Item3	Item3	Item3
Item1	Item1	Item1	Item2	Item2	Item2	Item3	Item3	Item3
Item1			Item2			Item3		

Na definição do *container* flexível através (**display:flex**) estabeleceu-se que os conteúdos seriam apresentados por linha (**flex-direction:row**) e que o alinhamento dos *items* seria feito a partir do topo do *container* (**align-items:flex-start**)

Os elementos aproveitam a largura disponível (400px), são dispostos com a mesma largura e pela ordem que surgem no HTML

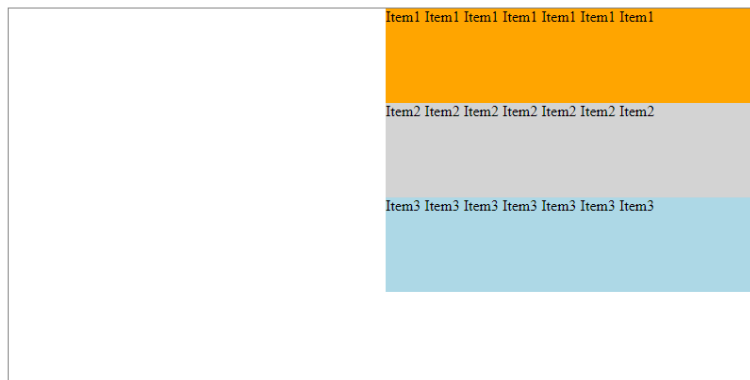
CSS Flexbox (layout)

```
<div id="container">
  <div id="d1">Item1 Item1 Item1 Item1 Item1 Item1 Item1 </div>
  <div id="d2">Item2 Item2 Item2 Item2 Item2 Item2 Item2 </div>
  <div id="d3">Item3 Item3 Item3 Item3 Item3 Item3 Item3 </div>
</div>
```

```
#d1{background-color: orange;}
#d2{background-color: lightgray;}
#d3{background-color: lightblue;}
```

```
#container{
  height:400px;
  width:800px;
  border: 1px gray solid;
  display:flex;
  flex-direction: column;
  align-items:flex-end;
}
```

```
#container div{
  height:100px;
  margin:0px;
  padding:0px;
  width: 400px;}
```

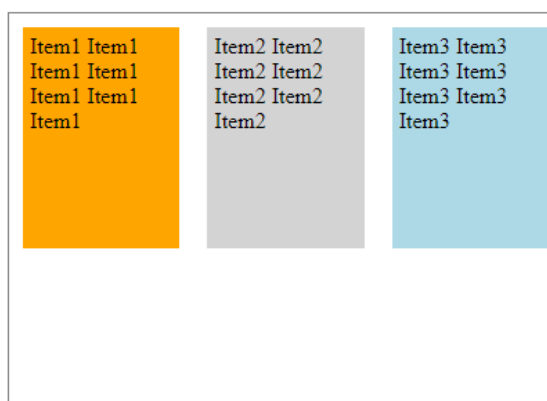


Tendo por base os mesmos elementos estruturais, é possível alterar de forma radical a forma como são apresentados, controlando apenas as propriedades do *flex-container*

CSS Flexbox (layout)

- O flex container permite que sejam aplicadas propriedades aos flex items:
 - Exemplo: neste caso podem ser definidos o *padding*, a *margin* e a *height* de cada flex item

```
#container div{
  height:150px;
  margin:10px;
  padding:5px;
}
```



CSS Flexbox (Layouts)

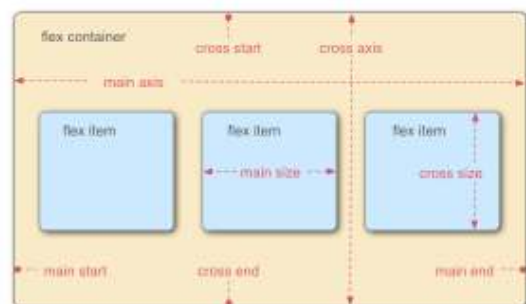
flex container

flex item

CSS Flexbox (layout)

■ Propriedades dos **flex items**

- **order**
 - permite reordenar os *flex items* num *container*
- **flex-grow**
 - estabelece quanto é que um item pode crescer se existir espaço livre no *flex container*
- **flex-shrink**
 - estabelece quanto é que um item pode reduzir caso não exista espaço no *flex container*
- **flex-basis**
 - define a dimensão inicial de um *flex item*
- **flex**
 - propriedade agregada (*flex*: *flex-grow flex-shrink flex-basis*)



■ Propriedades para os *flex items*:

■ {order: <integer>;}

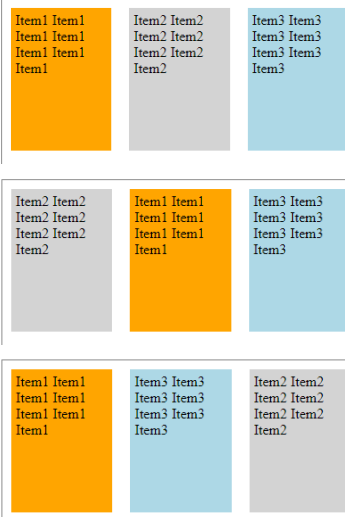
- Por defeito a ordem dos *items* segue a ordem do HTML, sendo que todos os *flex items* possuem o valor 0 (*order*).
- É possível controlar a posição de um item controlando o valor desta propriedade, um valor negativo coloca o *item* em primeiro lugar, pelo contrário um valor positivo posiciona o item depois de todos aqueles com um valor de *order* inferior.

```
<div id="container">
  <div id="d1">Item1...</div>
  <div id="d2">Item2...</div>
  <div id="d3">Item3...</div>
</div>
```

```
#container div{
  height:150px;
  margin:10px;
  padding:5px;
}
```

```
#d2{order:-1;}
```

```
#d2{order:1;}
```



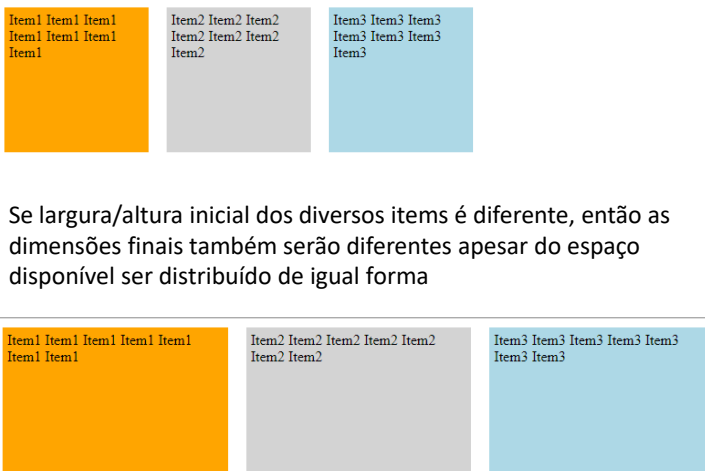
■ {flex-grow: <integer>;}

- Estabelece a capacidade de um item para crescer
- Aceita um valor que define a quantidade de **espaço disponível** que cada um dos *items* vai ocupar.
 - Se todos os *items* possuírem o valor 1 o espaço disponível é dividido de igual forma pelos diversos items
 - Caso contrário o item com o maior valor vai ocupar mais espaço do que os outros *items*.

```
#container{
  height:400px;
  width:800px;
  border: 1px gray solid;
  display:flex;
  flex-direction: row;
  align-items:flex-start;}

#container div{ width:150px;
  height:150px;
  margin:10px;
  padding:5px;}
```

```
#container div{flex-grow: 1;}
```



Se largura/altura inicial dos diversos items é diferente, então as dimensões finais também serão diferentes apesar do espaço disponível ser distribuído de igual forma

CSS Flexbox (layout)

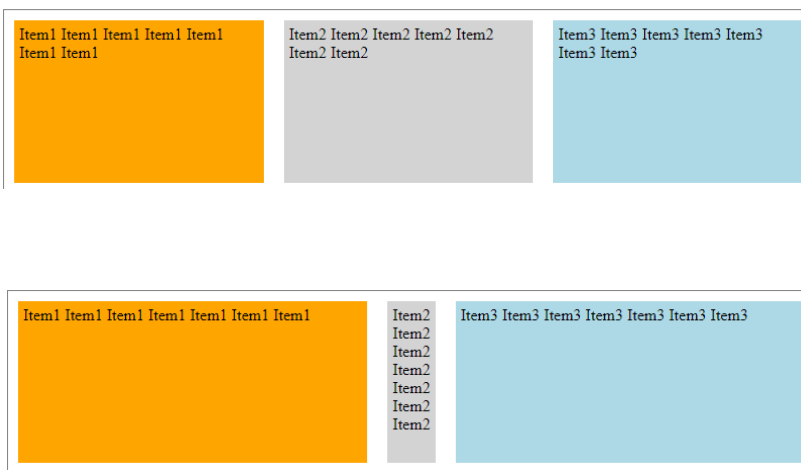
- {flex-shrink: <integer>;}
- Capacidade de um item para reduzir a sua dimensão, o valor inteiro define o grau de redução atribuído a esse elemento.
- Esta propriedade só se aplica em situação de **overflow** (largura items > largura do container)

```
#container{  
  height:400px;  
  width:800px;  
  border: 1px gray solid;  
  display:flex;  
  flex-direction: row;  
  align-items:flex-start;}
```

```
#container div{  
  height:150px;  
  margin:10px;  
  padding:5px;  
  flex-grow: 1;  
  flex-shrink: 1;  
  width: 400px;}
```

```
#container #d2{flex-shrink:6;}
```

Como a largura está definida e existe uma situação de *overflow*, a dimensão dos *flex-items* irá ajustar-se à largura do *flex container*

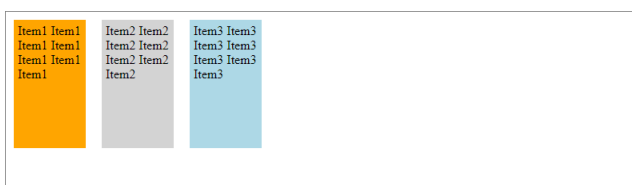


CSS Flexbox (layout)

- {flex-basis: <length> | auto;}
- Define a dimensão (% , em, ...) inicial de um flex item.

```
#container div{  
  height:150px;  
  margin:10px;  
  padding:5px;  
  /*flex-grow: 1;*/  
}
```

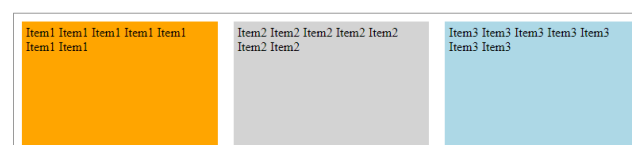
```
#container div{flex-basis:10%;}
```



- Se *flex-grow* definido o espaço livre é repartido pelos *flex-items*

```
#container div{  
  height:150px;  
  margin:10px;  
  padding:5px;  
  flex-grow: 1;  
}
```

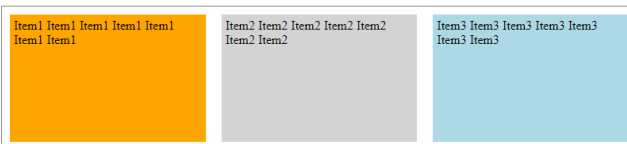
```
#container div{flex-basis:10%;}
```



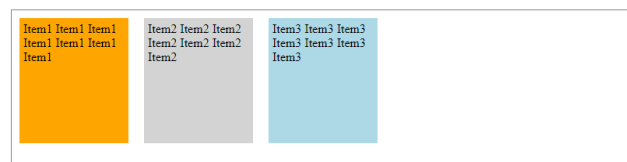
CSS Flexbox (layout)

- {flex-basis: <length> | auto;}
 - Se a largura do elemento não se encontrar definida, o valor **auto** ajusta ao conteúdo de cada um dos flex items
 - Caso a largura do *flex-item* tenha sido definida (*width*), {flex-basis:auto} não altera essa definição.

```
#container div{  
  height:150px;  
  margin:10px;  
  padding:5px;  
  /* flex-grow: 1; */  
  /* width: 130px;*/  
  
#container div{flex-basis:auto;}
```



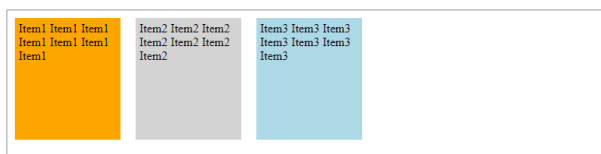
```
#container div{  
  height:150px;  
  margin:10px;  
  padding:5px;  
  /* flex-grow: 1; */  
  width: 130px;}  
  
#container div{flex-basis:auto;}
```



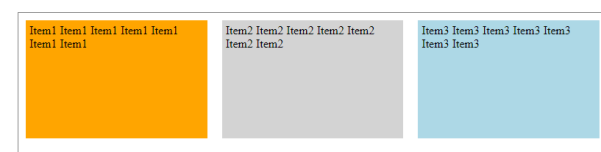
CSS Flexbox (layout)

- {flex: 0 1 auto}
- propriedade agregada para os valores *flex-grow*, *flex-shrink* (opcional) e *flex-basis* (opcional) .
- Os valores por defeito são (0 1 auto).

```
#container div{  
  height:150px;  
  margin:10px;  
  padding:5px;  
  width: 130px;}  
  
#container div{flex:0 1 auto;}
```



```
#container div{  
  height:150px;  
  margin:10px;  
  padding:5px;  
  width: 130px;}  
  
#container div{flex:1 1 auto;}
```



- **align-self**: auto || flex-start || flex-end || center || baseline || stretch
 - Permite o alinhamento de um *flex item*, mantendo o alinhamento previamente definido para os restantes *flex items*

```
#container div{  
  height:150px;  
  margin:10px;  
  padding:5px;  
  width: 130px;}
```

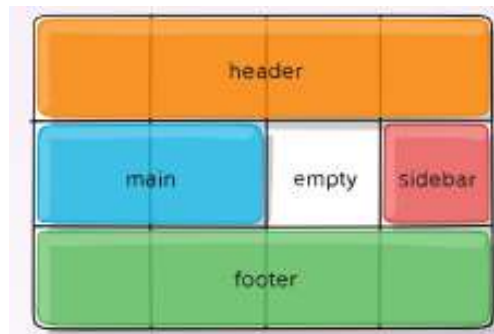
```
#container div{flex:1 1 auto;}
```

```
#d2{align-self:flex-end}
```



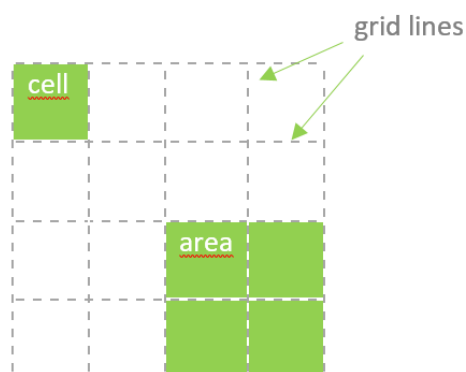
CSS Grid Layout

- A criação de um *grid container* é efetuada através de `{display: grid}`
 - Deve ser estabelecido o número de linhas/colunas do grid container
 - Todos os filhos **diretos** de um *grid container* tornam-se *grid items*
 - Associar cada *grid item* a uma área no *grid container*
 - pode ser criada uma grid dentro de uma outra grid e assim criar vários contextos de posicionamento



<https://css-tricks.com/snippets/css/complete-guide-grid/>

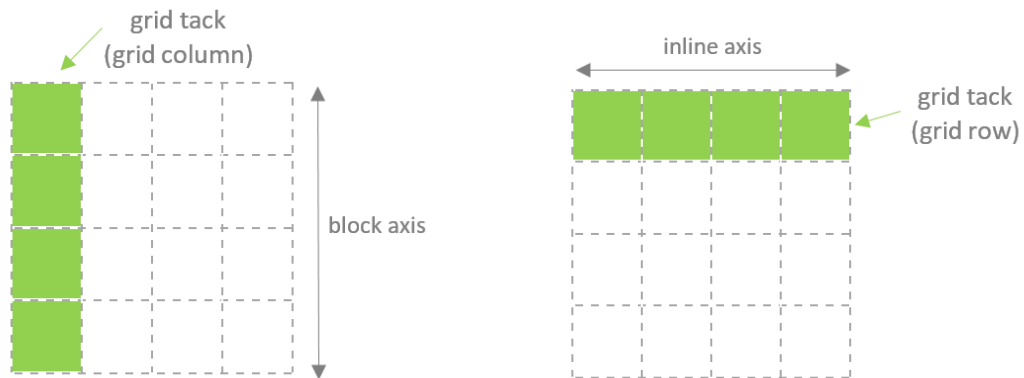
- CSS Grid
 - *line*
 - linhas horizontais e verticais que constituem a *grid*
 - *cell*
 - unidade mais pequena de uma *grid* a qual é delimitada por 4 *grid lines* adjacentes
 - *area*
 - área rectangular constituída por uma ou mais *grid cells* adjacentes



■ CSS Grid

■ track

- o espaço entre células adjacentes é um *grid track*, o qual pode ser um *grid column* ou *grid row*
 - *grid column* é vertical (*block axis*)
 - *grid row* é horizontal (*inline axis*)



CSS Grid Layout

Grid Container

display: grid
grid-template-rows
grid-template-columns
grid-template-areas

Definição de um Grid Container

- A declaração de um CSS grid é feito através da propriedade **display:grid**
 - **grid-template-rows**; **grid-template-columns** definem n.º e dimensões de linhas/colunas

```
#layout{display: grid;  
  grid-template-rows: 100px 400px 100px;  
  grid-template-columns: 200px 500px 200px;  
}  
  
</style>  
</head>  
<body>  
  
  <div id="layout">  
    <div id="one">One</div>  
    <div id="two">Two</div>  
    <div id="three">Three</div>  
    <div id="four">Four</div>  
    <div id="five">Five</div>  
  </div>
```

Definidas 3 linhas e 3 colunas.
O *template* por ser dividido no número de linhas e colunas definidos pelos valores diferentes atribuídos a **grid-template-rows**; **grid-template-columns**

Definição de um Grid Container

- A propriedade **grid-template-areas** permite identificar as várias *grid cells*
 - Importante!
 - Trata-se apenas da identificação das células uma vez que a grid e as respetivas dimensões das grid tracks são definidas através das propriedades **grid-template-rows**, **grid-template-columns**

```
#layout{display: grid;  
  grid-template-rows: 100px 400px 100px;  
  grid-template-columns: 200px 500px 200px;  
  grid-template-areas:  
    "header header header"  
    "ads main links"  
    "footer footer footer";  
}  
  
</style>  
</head>  
<body>  
  
  <div id="layout">  
    <div id="one">One</div>  
    <div id="two">Two</div>  
    <div id="three">Three</div>  
    <div id="four">Four</div>  
    <div id="five">Five</div>  
    <div id="six">Six</div>  
  </div>
```

Células são referenciadas por **linha**

header	header	header
ads	main	links
footer	footer	footer

- Existe ainda a propriedade **grid** que agrega as propriedades:
 - grid-template-rows
 - grid-template-columns
 - grid-template-areas
- não é muito utilizada uma vez que origina uma sintaxe mais complexa do que a utilização das propriedades individuais:

```
#layout{display: grid;
  grid:
    "header header header" 100px
    "ads main links"       400px
    "footer footer footer" 100px
    / 200px 500px 200px
}
```

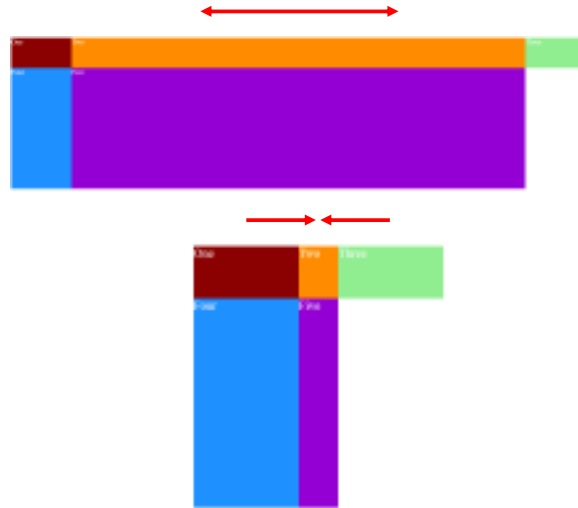
CSS Grid Layout

Grid Container

Unidades

- Na CSS grid as unidades a utilizar devem assegurar a flexibilidade necessária para um *responsive web design* (diversidade de dispositivos)
 - px (fixed); %
 - fr (*fractional unit*)
 - permite que os grid tracks se ajustem ao espaço disponível, expandam/retraiam de acordo com as variações do *viewport*.

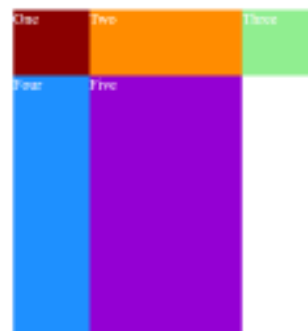
```
#layout{  
  display: grid;  
  grid-template-rows: 100px 400px 100px;  
  grid-template-columns: 200px 1fr 200px}
```



- As *fractional units* podem ser usadas para:
 - combinar áreas de dimensão variável com dimensão fixa (exemplo anterior)
 - combinar áreas de dimensão variável

```
#layout{  
  display: grid;  
  grid-template-rows: 100px 400px 100px;  
  grid-template-columns: 1fr 2fr 1fr}
```

A coluna central terá sempre o dobro da capacidade de ajustamento das colunas dos extremos



- A função `minmax()` permite limitar as dimensões de uma grid track
 - tornar essa *grid track* flexível mas dentro de limites pré-definidos

```
#layout{
  display: grid;
  grid-template-rows: 100px 400px 100px;
  grid-template-columns: 200px minmax(5em,15em) 200px}
```

- As dimensões de uma grid track também podem ser definidas com base nas dimensões do seu próprio conteúdo
 - `min-content`
 - o menor valor para não criar situações de overflow (ex: a palavra mais longa; a imagem mais larga, ...)
 - `max-content`
 - o valor máximo que permita conter o conteúdo sem wrapping. Não é adequado para parágrafos, o efeito pode ser imprevisível.

- As dimensões da grid podem ser definidas tirando partido de:
 - `repeat (number_of_repetitions, pattern)`

```
#six{background-color:darksalmon}

#layout{display: grid;
  grid-template-rows: 100px 400px 100px;
  grid-template-columns: repeat(2, 200px 1fr 200px)}

</style>
</head>
<body>
  <div id="layout">
    <div id="one">One</div>
    <div id="two">Two</div>
    <div id="three">Three</div>
    <div id="four">Four</div>
    <div id="five">Five</div>
    <div id="six">Six</div>
  </div>
```



CSS Grid Layout

Grid Container

Unidades

Posicionamento Grid Items

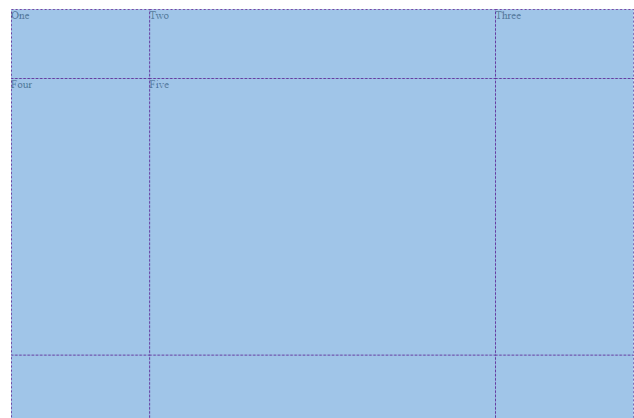
Posicionamento Grid Items

- Cada um dos *grid items* é atribuído de forma automática a cada uma das *grid cells*
 - por defeito essa atribuição é sequencial e feita por linha

```
#layout{display: grid;  
  grid-template-rows: 100px 400px 100px;  
  grid-template-columns: 200px 500px 200px}
```

```
div{color:white; font-size: 1.2em}  
#one{background-color:darkred}  
#two{background-color:darkorange}  
#three{background-color:lightgreen}  
#four{background-color:dodgerblue}  
#five{background-color:darkviolet}
```

```
</style>  
</head>  
<body>  
  
  <div id="layout">  
    <div id="one">One</div>  
    <div id="two">Two</div>  
    <div id="three">Three</div>  
    <div id="four">Four</div>  
    <div id="five">Five</div>  
  </div>
```



CSS Grid Layout

Definição de uma grid

Unidades

Posicionamento dos Grid Items

Linhas

Posicionamento Grid Items

■ Identificação das linhas

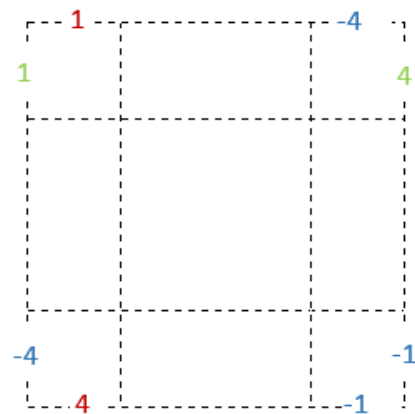
■ Identificação numérica

■ row line (1 – 4)

■ column line (1-4)

■ Contagem Inversa

- a contagem inicia-se na ultima linha sempre com o valor -1 e processa-se de forma decrescente (-2, -3, ...)



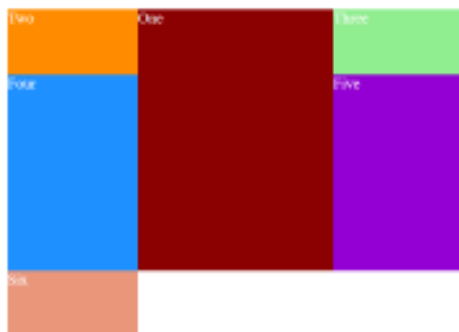
■ Identificação baseada em nomes [...]

```
#layout{display: grid;
  grid-template-rows: [header-start] 100px [content-start] 400px [footer-start] 100px;
  grid-template-columns: [adds] 200px [main] 500px [links] 200px}
```

Posicionamento Grid Items

- A alteração do posicionamento é efetuado a partir da identificação das linhas tendo por base as propriedades:

- grid-row-start
- grid-row-end
- grid-column-start
- grid-column-end



```
#layout{display: grid;
  grid-template-rows: 100px 300px 100px;
  grid-template-columns: 200px 300px 200px;
  grid-template-areas:
    "header header header"
    "ads main links"
    "footer footer footer"
}

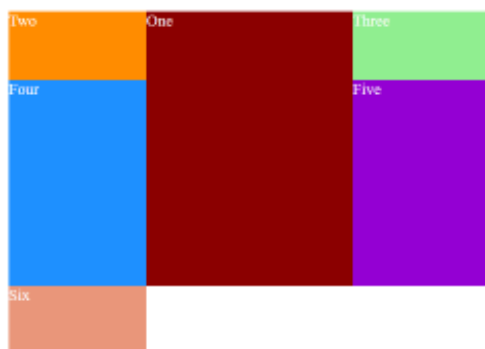
#one{
  grid-row-start: 1;
  grid-row-end:3;
  grid-column-start: 2;
  grid-column-end: 3;
}
```

`</style>`
`</head>`
`<body>`
`<div id="layout">`
`<div id="one">One</div>`

O item é posicionado, sendo que todos os outros reajustam de acordo com o espaço disponível
O posicionamento é feito pelo número/nome da linha

Posicionamento Grid Items

- O posicionamento anterior pode ser efetuado de forma mais condensada:
- grid-row: start line / end line
- grid-column: start line / end line



```
#layout{display: grid;
  grid-template-rows: 100px 300px 100px;
  grid-template-columns: 200px 300px 200px;
  grid-template-areas:
    "header header header"
    "ads main links"
    "footer footer footer"
}

#one{
  grid-row: 1 / 3;
  grid-column: 2 / 3;
}
```

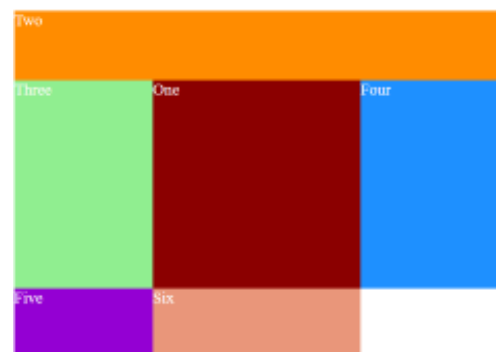
`</style>`
`</head>`
`<body>`
`<div id="layout">`
`<div id="one">One</div>`
`<div id="two">Two</div>`
`<div id="three">Three</div>`
`<div id="four">Four</div>`
`<div id="five">Five</div>`
`<div id="six">Six</div>`
`</div>`

Posicionamento Grid Items

- O posicionamento através de grid-area também pode ser feito com base na designação das áreas criadas através de **grid-template-areas**

```
#layout{display: grid;
  grid-template-rows: 100px 300px 100px;
  grid-template-columns: 200px 300px 200px;
  grid-template-areas:
    "header header header"
    "ads main links"
    "footer footer footer"}
#one{
  grid-area: main;
}
#two{
  grid-area: header;
}
</style>
</head>
<body>
  <div id="layout">
    <div id="one">One</div>
    <div id="two">Two</div>
    <div id="three">Three</div>
```

header	header	header
ads	main	links
footer	footer	footer

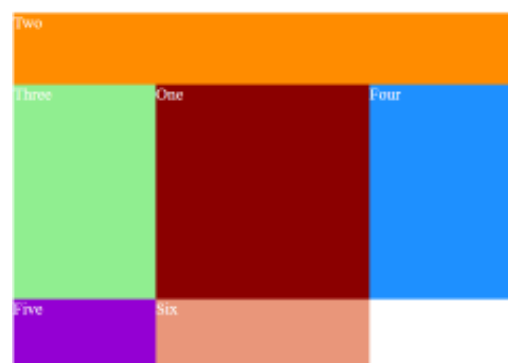


Posicionamento Grid Items

- A propriedade **grid-area** pode também ser aplicada com base em 4 grid lines as quais definem uma área:
 - `grid-area: row-start / column start / row-end / column-end;`

```
#one{
  grid-area: main;
}
#two{
  grid-area: 1 / 1 / 2 / 4;
}
</style>
</head>
<body>
  <div id="layout">
    <div id="one">One</div>
    <div id="two">Two</div>
    <div id="three">Three</div>
```

header	header	header
ads	main	links
footer	footer	footer



Posicionamento Grid Items

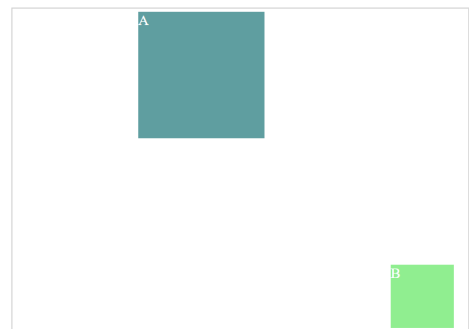
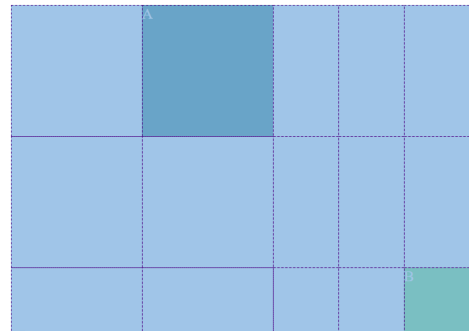
- Se o posicionamento for definido fora da grid inicial podem ser geradas automaticamente linhas/colunas para esse posicionamento ser possível
 - grid-auto-rows** e **grid-auto-columns** utilizadas para geração automática de *tracks* e aplicar à grid prédefinida

```
div{color:white; font-size: 1.2em}
#littlegrid{
  display: grid;
  grid-template-columns: 200px 200px;
  grid-template-rows: 200px 200px;
  grid-auto-columns: 100px;
  grid-auto-rows: 100px }

#a{
  grid-row: 1 / 2;
  grid-column: 2 / 3;
  background-color: cadetblue;}

#b{
  grid-row: 3 / 4;
  grid-column: 5 / 6;
  background-color:lightgreen; }

</style>
</head>
<body>
  <div id="littlegrid">
    <div id="a">A</div>
    <div id="b">B</div>
  </div>
```



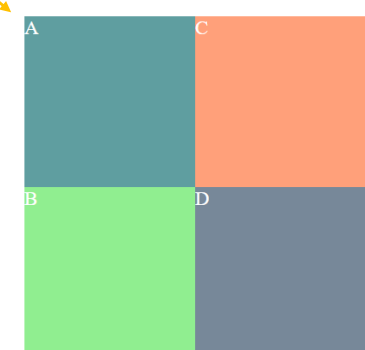
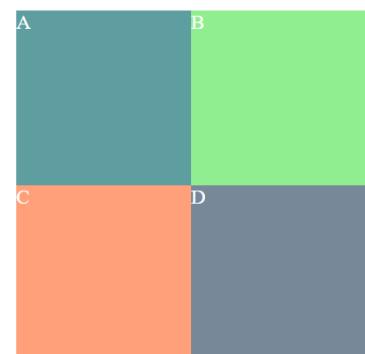
Posicionamento Grid Items

- A propriedade **grid-auto-flow** permite estabelecer a direção do posicionamento (*row (default); column*)

```
div{color:white; font-size: 1.2em}
#littlegrid{
  display: grid;
  grid-auto-flow: column;
  grid-template-columns: 200px 200px;
  grid-template-rows: 200px 200px;
  grid-auto-columns: 100px;
  grid-auto-rows: 100px }

#a{background-color: cadetblue;}
#b{background-color:lightgreen; }
#c{background-color:lightsalmon}
#d{ background-color:lightslategrey }

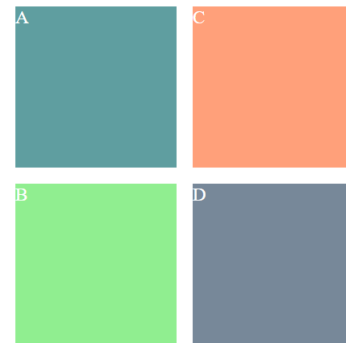
</style>
</head>
<body>
  <div id="littlegrid">
    <div id="a">A</div>
    <div id="b">B</div>
    <div id="c">C</div>
    <div id="d">D</div>
  </div>
```



Posicionamento Grid Items

- As propriedades **grid-row-gap/grid-column-gap** permitem estabelecer um espaço entre tracks
 - **grid-gap** permite agregar estes dois valores por linha e coluna

```
#littlegrid{
  display: grid;
  grid-auto-flow: column;
  grid-gap: 20px;
  grid-template-columns: 200px 200px;
  grid-template-rows: 200px 200px;
  grid-auto-columns: 100px;
  grid-auto-rows: 100px;
}
```



Posicionamento Grid Items

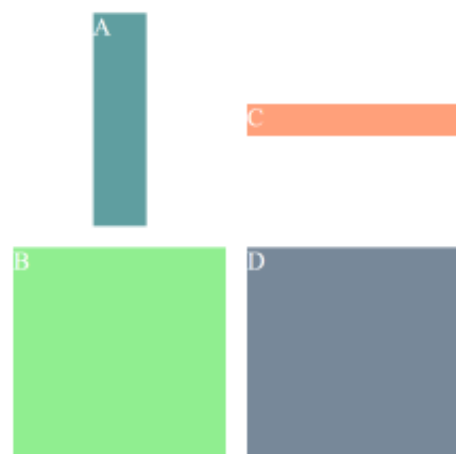
- Quando os items possuem dimensão inferior à grid que foi criada, podem ser alinhados (posicionados) tendo por base as propriedades:
 - **justify-self** (alinhamento horizontal)
 - **align-self** (alinhamento vertical)

```
#littlegrid{
  display: grid;
  grid-auto-flow: column;
  grid-gap: 20px;
  grid-template-columns: 200px 200px;
  grid-template-rows: 200px 200px;
  grid-auto-columns: 100px;
  grid-auto-rows: 100px;
}

#a{background-color: cadetblue;
  width:50px;
  justify-self: center;}

#b{background-color:lightgreen; }

#c{background-color:lightsalmon;
  height:30px;
  align-self:center}
```



Posicionamento Grid Items

- A propriedade ***justify-items*** permite o alinhamento horizontal para todos os items

```
div{color:white; font-size: 1.2em;
width:50px;
height:30px}

#littlegrid{
display: grid;
grid-auto-flow: column;
justify-items: start;
grid-gap: 20px;
grid-template-columns: 200px 200px;
grid-template-rows: 200px 200px;}
```



```
#littlegrid{
display: grid;
grid-auto-flow: column;
justify-items: end;
grid-gap: 20px;
...}
```

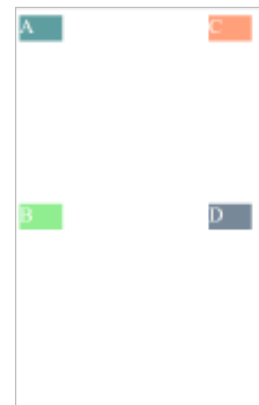


Posicionamento Grid Items

- A propriedade ***align-items*** permite o alinhamento vertical para todos os items

```
div{color:white; font-size: 1.2em;
width:50px;
height:30px}

#littlegrid{
display: grid;
grid-auto-flow: column;
align-items: start;
grid-gap: 20px;
grid-template-columns: 200px 200px;
grid-template-rows: 200px 200px}
```



```
#littlegrid{
display: grid;
grid-auto-flow: column;
align-items: end;
...}
```



Frameworks CSS (*layouts*)

Frameworks (*layout*)

■ Bootstrapp

- Instalação diretamente a partir de um *Content Delivery Network (CDN)*



<https://9official.com/2017/08/28/content-delivery-network-cdn/>

- Download rápido (possibilidade de *pre-cached files*) assim como controlo de versões

- <https://www.sitepoint.com/7-reasons-to-use-a-cdn/>

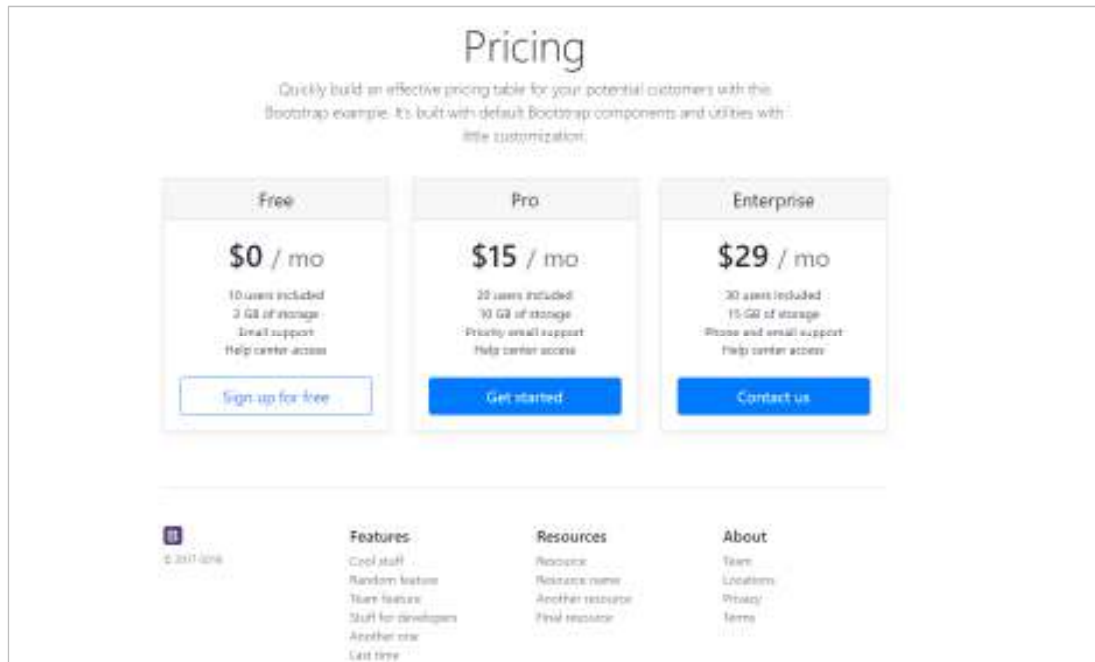
```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/css/bootstrap.min.css"
  integrity="sha384-/Y6pD6FV/Vv2HJnA6t+vs1U6fwYXjCFtcePhbNJ01yAFsXTsjBbfaDjzALeQsN6M"
  crossorigin="anonymous">
```

■ Bootstrap

All you need to do is learn the class names and apply them to your html markup. They'll do what they are designed to do.

<https://medium.freecodecamp.org/bootstrap-4-everything-you-need-to-know-c750991f6784>



<https://getbootstrap.com/docs/4.0/examples/>

Propriedades

display
visibility
float
position
formatação texto

Formatação de texto

■ *font*

Propriedade	Exemplo	Observações
font-family	body{font-family:Arial, sans-serif;}	Podem ser definidas famílias genéricas: serif/sans serif; monospace/proportional; cursive (escrita manual); fantasy Não há limite para o número de fontes definidas.
	body{font-family:Tahoma, Geneva, sans-serif; }	Deve-se começar por fontes específicas e terminar com uma genérica, de forma a garantir a correta interpretação pelo browser.
font-size	h1{font-size: 1.5em;}	Unidades relativas
	h1{font-size: 150%;}	Unidades absolutas
	h1{font-size: x-large;}	keywords: xx-small, small, medium*, large, x-large, ...

Formatação de texto

■ *font*

Propriedade	Exemplo	Observações
font-weight	h1 {font-weight: bold;}	Valores : normal*, bold, bolder, lighter, 100...900, inherit (herda o valor do elemento pai) Forma correcta de colocar elementos a bold, deve substituir o elemento HTML ;
font-style	h1 {font-style: italic;}	Valores : normal*, italic, oblique, inherit
font-variant	h1 {font-variant: small-caps;}	Valores: normal*, small-caps, inherit
font	h1 {font: italic bold inherit 1.5em Arial, sans-serif;}	Propriedade abreviada, onde a ordem dos valores é importante {font: style weight variant size font-family}

* - default value

Formatação de texto

■ Download de fontes

■ @font-face rule

- Evita a necessidade da fonte utilizada ter de estar previamente instalada no cliente.
- A fonte é instalada no web server e sempre que necessário é efetuado o seu download.

```
@font-face {  
    font-family: newFont;  
    src: url('myfont.ttf')  
}  
  
div{font-family:newFont;}
```

Formatação de texto

Propriedade	Exemplo	Observações
line-height	p {line-height: 2;} p {line-height: 2em;}	Define o espaçamento entre linhas. Valores: number (factor de escala), length measurement, percentage, normal*, inherit
text-indent	p#1 {text-indent: 2em;} p#1 {text-indent: 20%;}	Define a indentação de um parágrafo. Valores : length measurement (0*), percentage, inherit
text-align	p#1 {text-align: left;}	Alinhamento horizontal do texto. Valores: left*, right, center, justify, inherit
text-decoration	a {text-decoration: none;}	Permite criar sublinhados, linhas sobrepostas ao texto,...
text-transform	h1 {text-transform: none;}	Define o texto em maiúsculas, minúsculas, ... Valores: none*, capitalize (apenas a 1ª letra de cada palavra), lowercase, uppercase, inherit
letter-spacing	p {letter-spacing: 8px;}	Define o espaçamento entre letras. Valores: length measurement, normal*, inherit
word-spacing	p {word-spacing: 8px;}	Define o espaçamento entre palavras. Valores: length measurement, normal*, inherit

■ Múltiplas Colunas


```
p{
  background-color: orange;
  color:white;
  width: 600px;
  column-count: 3;
  font-size: 1.3em;
}
</style>
</head>
<body>
  <p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do ....
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud	exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu	fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum
--	--	--

Propriedades

display
visibility
float
position
formatação texto
cor

■ Propriedades: Formatação (cor)

Propriedade	Exemplo	Observações
		Valores: color value (RGB / hexadecimal); color name:
color	h1 {color: red;}	
	h1 {color: #FF0000;}	
	h1 {color: #F00;}	
	h1 {color: rgb(255,0,0);}	
http://www.w3schools.com/colors/colors_names.asp		
background-color	h1 {background-color: #F00;}	Valores: color value (RGB / hexadecimal); color name

■ Propriedades: Formatação (cor)

■ CSS color names (140)

- p {color: orange;}

■ RGB (red, green, blue)

- Decimal [0-255]
 - p {color: rgb(255,153,0);}

■ Hexadecimal # RRGGBB [0-9;A-F]

- p {color: #FF9900}
- p {color: #F90} (notação condensada quando a notação hexadecimal é composta por três pares de valores duplicados [utilizado por defeito em alguns editores de HTML/CSS])

Conversão decimal – hexadecimal:

dividir o número original por 16, o 1º dígito é o quociente e o 2º dígito o resto da divisão
 $200 = C8 = (16 \times 12 + 8)$

■ Existem vários *color palette generators* disponíveis:

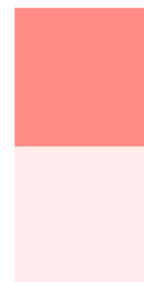
- <https://www.materialpalette.com/light-green/purple>
- <https://digitalsynopsis.com/design/website-color-schemes-palettes-combinations/>
- <http://htmlcolorcodes.com/resources/best-color-palette-generators/>



■ *alpha channel*

- grau de transparência na definição da cor
 - `h1 {color: #FF0000CC;}`
 - `rgba(255,0,0,0.8)`

```
.alpha{
  width:100px;height:100px;
  background-color: #ff6f69CC;
}
.alpha2{
  width:100px;height:100px;
  background-color: #ff6f6922;
}
</style>
</head>
<body>
  <div class="alpha"></div>
  <div class="alpha2"></div>
```



Propriedades

display
visibility
float
position
formatação texto
cor [opacity]

cor [opacity]

- A propriedade *opacity* é frequentemente utilizada
 - permite controlar o nível de transparência de um elemento

```
h1{color:darkorange}
#o1{opacity:1;}
#o2{opacity:0.5;}
#o3{opacity:0.25;}
</style>
</head>
<body>
  <h1 id="o1">Opacity</h1>
  <h1 id="o2">Opacity</h1>
  <h1 id="o3">Opacity</h1>
```

Opacity

Opacity

Opacity

Propriedades

display
visibility
float
position
formatação texto
cor [opacity]
imagens de fundo

Imagens de fundo

Propriedade	Observações
background-image	Imagem de fundo
background-repeat	Controla a repetição da imagem
background-position	Posição da imagem de fundo
background-attachment	Acompanha o deslocamento vertical dos conteúdos

- **{background-size: width height;}**
 - É possível definir várias dimensões para a imagem de fundo, eliminando a dependência das dimensões originais da imagem, admite vários valores:
 - **auto**: default (dimensões originais da imagem)
 - **length / percentage** (relativamente ao *container*)
 - **cover**: escala a imagem de modo a cobrir toda a área visível, algumas partes da imagem podem ficar invisíveis (as proporções da imagem original são mantidas)
 - **contain**: escala a imagem considerando a área visível (toda a imagem é visível). Caso seja necessário a imagem é repetida.

```
<style>
  body {color:white;
        background-image:url(back_Image.jpg);
        background-repeat:no-repeat;
        background-size:600px 100px;}
</style>
</head>
<body>
  <h1>Imagem de Fundo</h1>
```

Propriedades

display

visibility

float

position

formatação texto

cor [opacity]

imagens de fundo

listas

<i>Propriedade</i>	<i>Exemplo</i>	<i>Observações</i>
list-style-type	<code>ul {list-style-type:square;}</code>	Permite escolher o símbolo de uma lista. Valores: none, disc*, circle, square, ...
list-style-position	<code>li {background-color: #666;}</code> <code>ul {list-style-position: outside;}</code>	Define se o símbolo da lista se encontra no interior ou exterior do background aplicado aos elementos da lista Valores : inside, outside, inherit
list-style-image	<code>ul {list-style-image: url(/image.png)}</code> <code>list-style-position: outside;}</code>	Criar novos símbolos de lista (bullets) Valores: url, none*, inherit

* - default value

Propriedades

display

visibility

float

position

formatação texto

cor [opacity]

imagens de fundo

listas

tabelas

■ Propriedades específicas para tabelas

Propriedade	Observações
<i>border-collapse</i>	Permite definir a separação entre células (borders separados, ou apenas um único border a separar células adjacentes). Valores: separate, collapse, inherit
<i>border-spacing</i>	Permite definir o espaçamento entre células. Valores: length, inherit
<i>empty-cells</i>	Permite definir se as células vazias são visíveis Valores: show, hide, inherit

```
<style>
  table{
    border-style:solid;
    border-width:1px;
    border-color:#900;
    border-collapse: separate;
    border-spacing:10px;
    empty-cells:hide;}

  th, td {border-style:solid;
    border-width:1px;
    border-color:#900;
    width:100px;
    text-align:center;}
</style>
```

```
<table>
  <tr><th>(1,1)</th>
    <th>(1,2)</th>
  </tr>
  <tr><td>(2,1)</td>
    <td>(2,2)</td>
  </tr>
  <tr><td>(3,1)</td>
    <td></td>
  </tr>
</table>
```

(1,1)	(1,2)
(2,1)	(2,2)
(3,1)	

Propriedades

display
visibility
float
position
formatação texto
cor [opacity]

imagens de fundo
listas
tabelas
formulários

formulários

- Uma formatação correta dos elementos de um formulários é determinante para assegurar a sua usabilidade (objetivo principal)
 - Existem determinadas propriedades que só se aplicam a campos específicos

Elementos	Propriedades
text inputs (text, password, email, search, tel, url)	<i>width, height, background-color, background-image, border, border-radius, margin, padding, box-shadow, color, ...</i>
textarea	<i>line-height, resize</i>
inputs (submit, reset, button)	<i>width, height, border, background-color, margin, padding, box-shadow, ...</i>
select	<i>width, height, color, background-color ...</i>
fieldset / legends	<i>border, background-color, margin, padding, ...</i>

Propriedades

display
visibility
float
position
formatação texto
cor [opacity]

imagens de fundo
listas
tabelas
formulários
transformações 2D/3D

Transformações 2D

- Propriedade com múltiplos valores: http://www.w3schools.com/cssref/css3_pr_transform.asp
 - `seletor{ transform: rotate(deg)}`

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title></title>
  <style>
    div {width:200px;
        height:100px;
        background-color:darkorange;
        color:white;
        position:absolute;left:50px;top:100px;
        text-align:center;
        transform:rotate(30deg);
    }
  </style>
</head>
<body>
  <div>transform:ROTATE(</div>
</body>
</html>
```



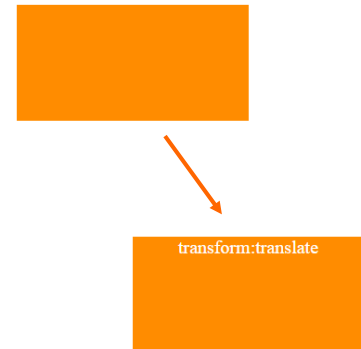
Por defeito, a imagem roda sobre o seu centro.

A propriedade *transform-origin* permite alterar o centro de rotação.

Transformações 2D

- `seletor{ transform: translate(xx,yy)}`
- A referência para o movimento de translação é a posição original do objecto

```
<!DOCTYPE html>
<html>
<head>
  <style>
    div {width:200px;
        height:100px;
        background-color:darkorange;
        color:white;
        position:absolute;left:50px;top:100px;
        text-align:center;}
    #d2 {transform:translate(100px,200px);}
  </style>
</head>
<body>
  <div></div>
  <div id="d2">transform:translate</div>
</body>
</html>
```



Transformações 2D

- `seletor{ transform: scale(xx,yy)}`

```
<!DOCTYPE html>
<html>
<head>
  <style>
    div {width:60px;
        height:20px;
        background-color:darkorange;
        color:white;
        position:absolute;left:50px;top:100px;
        text-align:center; }
    #d2 {left:150px;
        transform:scale(2,3);}
  </style>
</head>
<body>
  <div></div>
  <div id="d2">scale</div>
</body>
</html>
```



■ `seletor{transform: rotate X}; seletor{transform: rotateY}`

```
...
    .dft{ background-color: darkorange;
          color:white;
          font-size: 2em;
          width:50px;
          height:100px;
          margin:10px;
          text-align: center}

    .r3d{ transform: rotateX(180deg);}
</style>
</head>

<body>

    <div class="dft">1</div>
    <div class=" dft r3d">1</div>
    ...
```



Propriedades

display

visibility

float

position

formatação texto

cor [opacity]

imagens de fundo

listas

tabelas

formulários

transformações 2D/3D

transições

■ Suavização de alterações entre estados diferentes

- Definir uma transição envolve:
 - Propriedade a alterar (obrigatório!)
 - Duração (obrigatório!)
 - A forma como se processa a aceleração da transição
 - A eventual existência de uma pausa antes de iniciar a transição.

Propriedade	Exemplo	Observações
transition-property	{transition-property: width;}	Algumas propriedades que podem ser animadas: background-color; background-position; height, width, font-size; font-weight;...
transition-duration	{transition-duration: 1s;}	Os valores podem ser definidos em segundos (s) ou milisegundos (ms)
transition-timing-function	{transition-timing-function: ease;}	ease (início lento, rápido, final lento); linear (a velocidade mantém-se do início ao fim); ease-in (início lento, final rápido); ease-out (início rápido, final lento); ease-in-out (início lento, rápido, final lento). Existem outras opções que no entanto são menos utilizadas.
transition-delay	{transition-delay: 0.3s;}	Os valores podem ser definidos em segundos (s) ou milisegundos (ms)
transition	{transition: width 1s ease-in-out 0.3s;}	{transition: property duration timing-function delay}

```
<!DOCTYPE html>
<html>
<head>
  <style>
    div {width:100px;
        height:100px;
        background-color:orange;
        transition:width 3s; }
    div:hover {width:300px;}
  </style>
</head>
<body>
  <div></div>
</body>
</html>
```

Transição width Elemento div



Uma vez o rato sobreposto ao elemento div, efectua-se a transição da posição inicial para a posição final (duração 3s)

Transição width Elemento div



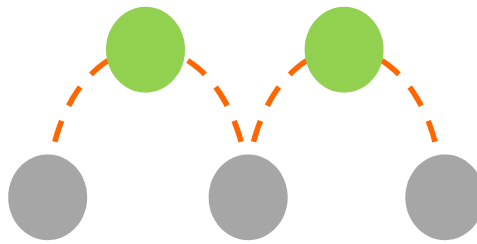
Propriedades

display
visibility
float
position
formatação texto
cor [opacity]

imagens de fundo
listas
tabelas
formulários
transformações 2D/3D
transições
animação

Animação

- Baseada em *key frames* (*keyframe animation*)
 - Transições são animações baseadas em 2 *key frames* (inicial, final)
 - Animações mais complexas requerem a especificação de um maior número de *key frames*



Animação

Keyframes

@keyframes nomeAnimação { ... }

Parâmetros

▪ Definição de *key frames*

nome da animação (obrigatório)

```
@keyframes quadrados {  
  0% {background-color:red; left:0px; top:0px}  
  25% {background-color:yellow; left:200px; top:0px}  
  50% {background-color:blue; left:200px; top:200px}  
  75% {background-color:green; left:0px; top:200px}  
  100% {background-color:red; left:0px; top:0px}  
}
```

Definição das *key frames* (obrigatório)
(neste exemplo foram definidas 5 *key frames*)

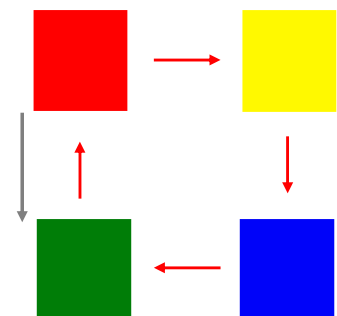
▪ Definir os parâmetros da animação

- *animation-name* (**obrigatório!**)
- *animation-duration*
 - Tempo, definido em segundos, que uma animação demora
 - o valor por defeito é o 0 (sem animação)
 - os valores negativos são tratados como 0
- *animation-timing-function*
 - *ease* : Valor por defeito. A animação começa lenta, depois rápido, antes de terminar lento
 - *linear* : Velocidade constante ao longo da animação
 - *ease-in* : Animação inicia lenta
 - *ease-out* : Animação termina lenta
 - *ease-in-out* : Animação com um começo e um final lentos
 - *cubic-bezier(x1, y1, x2, y2)* : Controlo mais preciso da velocidade da animação

- *animation-delay*
 - define o tempo de intervalo (definido em s) entre a ocorrência do evento e o início da animação
- *animation-iteration-count*
 - define o número de vezes que a animação é executada
 - Infinite: animação é executada de forma ininterrupta
- *animation-direction*
 - normal
 - valor por defeito. A animação é sempre reproduzida na mesma direção
 - alternate
 - executa as iterações ímpares na direção normal e as iterações pares na direção inversa
- ...

- Animação
 - Parametrização da animação e associação com *key frames* definidas

```
<!DOCTYPE html>
<html>
<head>
  <style>
    div {width:100px;
          height:100px;
          background-color:red;
          position:relative;
          animation:quadrados 5s infinite alternate;}
    @keyframes quadrados {
      0% {background-color:red; left:0px; top:0px}
      25% {background-color:yellow; left:200px; top:0px}
      50% {background-color:blue; left:200px; top:200px}
      75% {background-color:green; left:0px; top:200px}
      100% {background-color:red; left:0px; top:0px}
    }
  </style>
</head>
<body>
  <h2>Animação Quadrados</h2>
  <div></div>
</body>
</html>
```



Responsive Web Design

Responsive Web Design

- Permite que uma aplicação seja utilizada de forma adequada independentemente do dispositivo que está a ser utilizado para aceder aos conteúdos
 - Todos os dispositivos acedem o mesmo HTML, localizado no mesmo URL;
 - São aplicados estilos diferentes, de acordo com o dispositivo, de forma a redimensionar componentes e otimizar a usabilidade (interação com o utilizador)



<https://wpdune.com/creating-a-responsive-design-ui-guidelines-you-shouldnt-miss/>

■ Três componentes principais:

■ layout flexível (*flexible grid*)

- As dimensões não podem ser estáticas, têm de ajustar (reduzir/aumentar) de acordo com o espaço disponível.

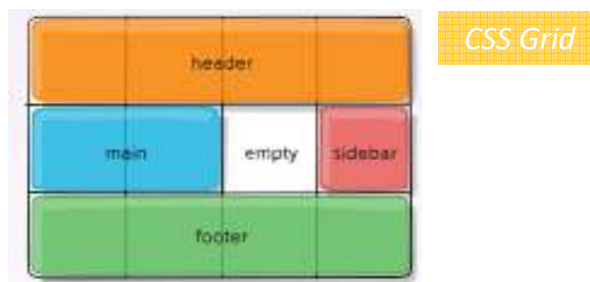
■ Imagens flexíveis

- Imagens com capacidade de sofrerem um efeito de escala (redução da dimensão) de acordo com a alteração do layout.

■ CSS media queries

- método que permite a aplicação de estilos de acordo com o dispositivo em que o *site* vai ser visualizado.

■ Layout flexível



Flexbox Container



Fluid layout



■ Imagens/media Flexíveis

- A imagem acompanha a redução do layout sendo que na situação contrária a sua dimensão original é o limite para a sua visualização.

```
img{max-width: 100%;}
```

- A propriedade *max-width* estabelece a largura máxima de um elemento
- Se for definida em %, a largura do elemento é **diretamente indexada às dimensões do respetivo container** até à dimensão original da imagem (\neq width:100%).
 - O elemento ajusta-se automaticamente à variação da largura do respetivo *container*
- Também aplicável a outro tipo de média

```
img, object, video {max-width:100%;}
```

■ CSS media queries

- Permite a aplicação de estilos de acordo com meio usado (*media type*) para visualização dos conteúdos web assim como em função das suas características (*media features*)

■ CSS media queries

■ media types

- screen, print, all, ... (opcional)

■ media features

- width; height; device-width; ...

■ CSS media queries

■ media features

Media feature	Observações
<i>width</i>	largura do viewport
<i>height</i>	altura do viewport
<i>orientation</i>	portrait/landscape
<i>aspect-ratio</i>	razão entre a largura e a altura do viewport
...	...

- Algumas propriedades podem ser testadas para valores limite, utilizando para tal prefixos:

■ *min, max*

- min-width; max-width
- min-height; max-height

■ CSS media queries

- Efetuados diretamente na folha de estilos (a forma mais comum):

```
@media screen and (min-width:480px){  
  
    /* CSS Rules */  
}  
  
@media screen and (min-width:480px) and (orientation:landscape){  
  
    /* CSS Rules */  
}
```

- Em alternativa ser incorporados no html, incorporando os *media queries* na tag **<link>** através do atributo **media**

```
<head>  
  <link href="geral.css" rel="stylesheet">  
  <link href="colunas.css" rel="stylesheet" media="screen and (min-width:480px)">  
</head>
```

Responsive Web Design

- Quando se utilizam *@media rules* a **ordem** das regras é muito importante (para seletores iguais prevalecem as ultimas regras a ser definidas):
 - Estratégia:
 - Especificam-se as regras aplicadas por defeito aos elementos (*baseline rules*)
 - Algumas dessas regras vão ser posteriormente substituídas por outras regras inseridas nas *@media* de forma a otimizar o conteúdo para determinados viewports.
 - A definição de *media queries* é baseada do conceito de *mobile-first*, ou seja:
 - Começa-se por definir os layouts para os dispositivos mais pequenos;
 - À medida que o espaço de visualização aumenta, são aplicados novos estilos para novas possibilidades de visualização.

```
@media screen and (min-width:480px){  
    /* CSS Rules */  
}  
  
@media screen and (min-width:768px){  
    /* CSS Rules */  
}
```

Responsive Web Design

- Definição das *@media queries*
 - Escolha dos **breakpoints** (largura estabelecida no *media query* para definir novos estilos)
 - <http://responsivedesign.is/develop/browser-feature-support/media-queries-for-common-device-breakpoints>
 - <http://css-tricks.com/snippets/css/media-queries-for-standard-devices/>

```
/* ----- iPhone X ----- */  
  
/* Portrait and Landscape */  
@media only screen  
    and (min-device-width: 375px)  
    and (max-device-width: 812px)  
    and (-webkit-min-device-pixel-ratio: 3) {  
  
}
```

■ CSS Pixel

- Corresponde ao pixel definido (abstração) nas declarações CSS

- width:200px; padding:10px;...

- <https://www.w3.org/TR/CSS2/syndata.html#length-units>

■ pixel (*device*)

- número de pixels existentes no dispositivo (hardware pixels - resolução)

- *landscape mode* (ex: iphone6 1334x750 px)

■ *device pixel ratio*

- A relação que existe entre os pixels (device) e os pixels CSS.

- Exemplo: 1 CSS pixel corresponde a 4 *device pixels* (DPR = 2)



<http://blog.popupdesign.com.br/desenvolvimento-responsivo-e-viewport/>

resolution

Pixel (CSS)
device width / device height

Common Smartphones values					
name	phys. width	phys. height	CSS width	CSS height	pixel ratio
Apple iPhone X	1125	2436	375	812	3
Apple iPhone 6+, 6s+, 7+, 8+	1080	1920	414	736	3
Apple iPhone 7, iPhone 8	750	1334	375	667	2
Apple iPhone 6, 6s	750	1334	375	667	2
Apple iPhone 5	640	1136	320	568	2
Apple iPhone 4	640	960	320	480	2
Apple iPhone 3	320	480	320	480	1
Apple iPod Touch	640	1136	320	568	2
LG G5	1440	2560	360	640	4
LG G4	1440	2560	360	640	4

<http://mydevice.io/devices/>

■ Variação da Janela de Visualização (viewport)

■ `<meta name="viewport" ... />`

- A especificação do *viewport* indica ao *browser* que deve ser aplicado um factor de escala à página para esta se ajustar às dimensões do dispositivo (screen)
- Deve ser incluído em todos os documentos html (responsive)

Property	Description
width	The width of the virtual viewport of the device.
device-width	The physical width of the device's screen.
height	The height of the "virtual viewport" of the device.
device-height	The physical height of the device's screen.
initial-scale	The initial zoom when visiting the page. 1.0 does not zoom.
minimum-scale	The minimum amount the visitor can zoom on the page. 1.0 does not zoom.
maximum-scale	The maximum amount the visitor can zoom on the page. 1.0 does not zoom.
user-scalable	Allows the device to zoom in and out. Values are yes or no.

■ O *viewport* necessita de ser declarado apenas uma vez

“A `<meta>` viewport element gives the browser instructions on how to control the page's dimensions and scaling”

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

■ ***content*** = ***“width=device-width,”***

- Largura do *viewport* igual à largura do dispositivo

■ ***content*** = ***“width=device-width, initial-scale=1”***

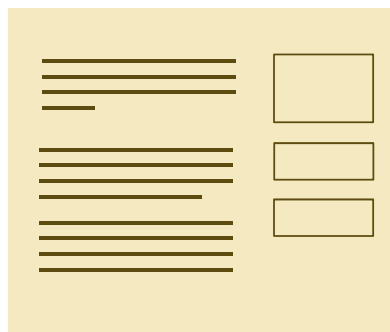
- Garante o nível de zoom inicial quando é feito o download da página
- Evita que seja visualizado apenas uma parte do conteúdo inicial

Responsive Web Design

Exemplo mobile safari
renderizado: 980px



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



screen: 320px



renderizado: 320px

Todo o espaço disponível é aproveitado, uma vez que o viewport é definido como sendo igual à largura do dispositivo

screen: 320px

Responsive Web Design

- Sem definição da meta tag, considerado por defeito um viewport de 980px largura
 - exemplo: iPhone 6/7



A imagem é renderizada a 980px e depois ajustada para a dimensão do dispositivo (neste caso é desperdiçado cerca de 1/3 do espaço disponível)

- Com ajuste do viewport

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



A imagem é renderizada considerando a largura do dispositivo

(optimização do espaço disponível)

■ Testing

■ Emuladores

- <http://www.mobilexweb.com/emulators>
- <https://crossbowseresting.com/>

■ Dispositivos reais

- <https://opendevicelab.com/>
- <https://www.foolproof.co.uk/device-lab/>



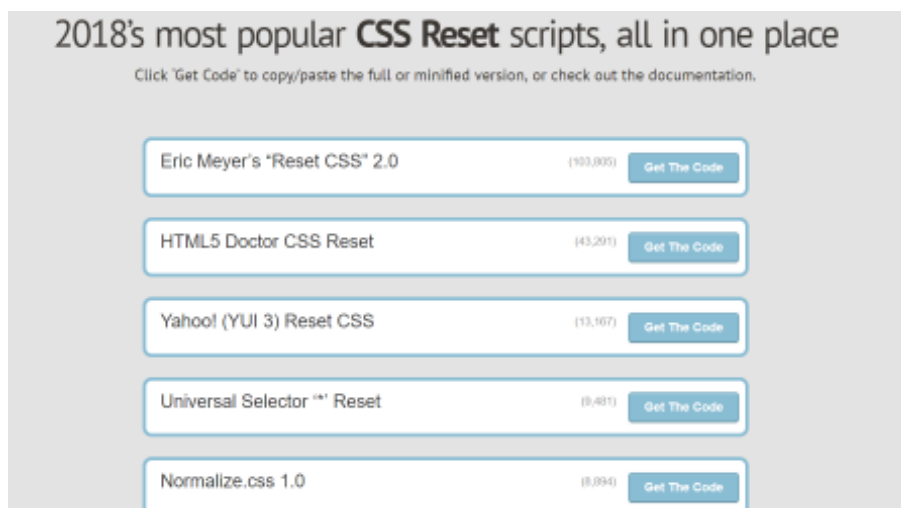
<https://devicelab.fi/>

CSS Reset

CSS Reset

■ CSS reset

- Aplicação de um conjunto de regras CSS que inviabilize a aplicação de estilos por defeito (browsers) de forma a tornar o ponto de partida tão neutro quanto possível.
- Existem disponíveis vários tipos de CSS *reset*



<http://cssreset.com/>

CSS Reset

- O CSS *reset* deve ser copiado para o início da CSS de forma a garantir que é efetuado de forma correta.
- Forma simples de evitar inconsistências de formatação dos diversos browsers

```
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}

/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
    display: block;
}
body {
    line-height: 1;
}
ol, ul {
    list-style: none;
}
blockquote, q {
    quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
    content: '';
    content: none;
}
table {
    border-collapse: collapse;
    border-spacing: 0;
}
```

<http://meyerweb.com/eric/tools/css/reset/>



Normalize.css

A modern, HTML5-ready alternative
to CSS resets

*“Normalize.css makes browsers render all elements more consistently and in line with modern standards. **It precisely targets only the styles that need normalizing.**”*

```
/*! normalize.css v8.0.1 | MIT License | github.com/necolas/normalize.css */

/* Document
   ========================================================================= */

/**
 * 1. Correct the line height in all browsers.
 * 2. Prevent adjustments of font size after orientation changes in iOS.
 */

html {
  line-height: 1.15; /* 1 */
  -webkit-text-size-adjust: 100%; /* 2 */
}

/* Sections
   ========================================================================= */

/**
 * Remove the margin in all browsers.
 */

body {
  margin: 0;
}
```

<https://necolas.github.io/normalize.css/8.0.1/normalize.css>