Aula 02

- CSS Iniciante e Intermédio

No final desta aula, serás capaz de:

- Definir o que é CSS e demonstrar três formas diferentes de incluír CSS na nossa página HTML
- Usar selectores para formatar elementos HTML
- Explicar como a especificidade joga um papel importante no uso de selectores

Introdução ao CSS

O HTML é basicamente a estrutra que sustenta uma página web, *o css é responsável por estilizar.* O que vem a ser CSS e como podemos usar este recurso nos nossos projectos?

CSS (**C**ascading **S**tyle **S**heets), é uma forma de descrever as regras de estilo que nós gostariamos de associar aos nossos elementos HTML. Usando CSS nós podemos alterar a cor, fonte (tipo de letra), margens, alinhamento de texto, e muito mais!. Clique neste link para aprender sobre o historial de CSS.

Três formas de usar CSS no HTML

1. Inline styling (*Estilização no elemento)* é processado com o uso do atributo **style** em uma tag **HTML**, veremos que este método tem alguns efeitos controversos.

2. <**style**></**style**> (Estilização no documento) é processado com o uso da tag <**style**></**style**> , sendo que o conteúdo da tag serão regras de estilização.

3. External stylesheets (Estilização externa) esta forma é a mais comum e também a mais recomendada.

External Stylesheet (Estilização Externa)

Para utilizar estilização externa, nós recorremos ao uso da tag <**link**>:

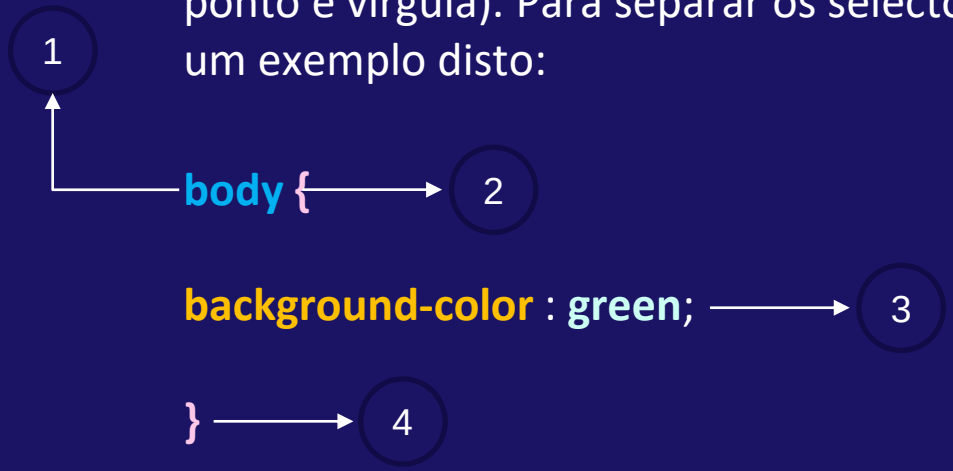
<**link** **rel**=“stylesheet” **href**=“style.css”>

Que deve ser posicionada no cabeça do documento HTML, muito especificamente, como filho da tag **head**.

```
<head>
        ...
        <link rel=“stylesheet” href=“style.css”>
        ...
</head>
```

Sintaxe e Selectores do CSS

Todas as regras CSS são compostas de um selector (um selector é a forma como encontramos um ou mais elementos HTML) e um par de chaves {}. Dentro das chaves, nós especificamos as propriedades e valores; nós separamos as propriedades e os valores com : (dois pontos) e no final ; (um ponto e vírgula). Para separar os selectores, nós usamos (vírgulas). Vejamos um exemplo disto:



Sintaxe e Selectores do CSS

```
body    Selector  {
        background-color  :  green;
        Propriedade          Valor
}
```

Nesta regra, nós estamos a selecionar o elemento <**body**> e a aplicar sobre este a formatação background-color (que deverá alterar a cor de fundo da nossa página para verde).

Esta técnica funciona, mas é muito GENÉRICA (talvez, queiramos ter *apenas* certas partes da nossa página com fundo verde e não toda página). Para sermos mais especificos, nós podemos utilizar atributos como **class** e **id** nos nossos elementos HTML e de forma individual definir regras de estilização.

Sintaxe e Selectores do CSS

Nós falaremos um pouco mais sobre os atributos **class** e **id** nos próximos capítulos. Por agora, podemos pensar neles como sendo utilidades que tornam mais simples o trabalho de selecionar elementos HTML nas nossas regras CSS.

A grande diferença entre **id** e **class** é que um **id** deve ser exclusivo para um elemento em particular, ao passo que uma **class** pode albergar muitos elementos.

Selector Descendente

Selector Descendente

O Selector descendente encontra todos os elementos que são descendentes de um elemento específico. O Selector abaixo, vai encontrar todo os elementos **p** que forem filhos do elemento **footer** (Os selectores são lidos da direita para esquerda).

```
footer p {
        background-color : green;
}
```

```
<footer>
        <header>
                <p>texto1</p>
        </header>
        <section>
                <p>texto2</p>
        </section>
</footer>
```

Selector Adjacente

Selector Adjacente

O Selector adjacente encontra todos os elementos *directamente* adjacentes a um outro elemento especificado. O Selector abaixo encontrará todos os elementos **h4** que estão directamente adjacentes ao elemento **h1.**

```
h1+h4 {
        background-color : green;
}
```

```
<section>
        <h1>Olá Escola!</h1>
        <h4>Olá turma!</h4>
        </h4>Olá colega</h4>
</section>
```

Selector Descendente directo

Selector descendente directo

Este selector é normalmente confundido com o selector descendente. O Selector descendente directo aplica-se *somente* aos elementos com descendencia directa.

```
section > p {
        color : green;
}
```

```
<section>
        <p>Olá Escola!</h>
        <div>Olá turma!
                <p>Parágrafo</p>
        </div>
        </p>Olá colega</p>
</section>
```

Selector atributo

Selector atributo

Este selector encontra um elemento com base no atributo do elemento. O Selector abaixo, vai encontrar todos os elementos **a** que tenham um atributo **href** configurado com o valor “#” (Regarrega a página).

```
a[href=“#”] {
        font-size : 20px;
}
```

```
<ul>
        <li>
                <a href=“#”>link1</a>
        </li>
        <li>
                <a href=“#”>link2</a>
        </li>
</ul>
```

Selector primeiro ou último filho

Selector primeiro ou último filho (first-child / last-child)

Este selector encontra todos os elementos que sejam os primeiros ou os últimos filhos de seus pais. O Selector abaixo vai encontrar todos os elementos **li** que são os primeiros filhos dos seus elementos pais.

```
li:first-child {
        background-color : red;
}
```

```
<ul>
        <li>
                <a href="#">link1</a>
        </li>
        <li>
                <a href="#">link2</a>
        </li>
</ul>
```

Selector primeiro ou último filho

Selector primeiro ou último filho (first-child / last-child)

Este selector encontra todos os elementos que sejam os primeiros ou os últimos filhos de seus pais. O Selector abaixo vai encontrar todos os elementos **li** que são os primeiros filhos dos seus elementos pais.

```
li:last-child {
        background-color : red;
}
```

```
<ul>
        <li>
                <a href="#">link1</a>
        </li>
        <li>
                <a href="#">link2</a>
        </li>
</ul>
```

Selector enésimo filho

Selector enésimo filho

Este selector encontra todos os elementos que sejam os filhos cuja a posição seja *n* na lista de filhos do elemento pai

```
p:nth-child(4) {
        background-color : red;
}
```

```
<section>
        <p>Parágrafo 1</p>
        <p>Parágrafo 2</p>
        <p>Parágrafo 3</p>
        <p>Parágrafo 4</p>

</section>
```

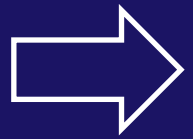
Outros Selectores

Os Selectores que acabamos de estudar, não são os únicos. Consulte esta lista para saber mais. O CSS tem efeito cascata, aliás, o C de CSS significa “Cascading” ou cascata em português. É desta forma que as regras mais abaixo em um ficheiro .css serão as que vão prevalecer. Vejamos um exemplo para efeitos de esclarecimento.

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

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

Outros Selectores

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

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



Esta regra tem precedência

Se estas duas regras foram declaradas no mesmo ficheiro .css, a última tem precedência sobre a primeira. Assim declara o efeito cascata do css.

Exercícios

Selecione um conjunto de alunos (um por vés) para demonstrar o uso dos diversos selectores CSS que aprendemos durante a aula.

No final desta aula, serás capaz de:

- Definir o que é especificidade
- Entender a diferença entre usar atributo elemento, atributo classe, atributo id, estilização interna (inline styling) e estilização !important
- Comparar e constrastar ids e classes

Especificidade

Nós aprendemos até agora que o CSS é usado através da selecção de elementos HTML e seus atributos e posteriormente aplicar sobre o(s) elemento(s) selecionado(s) uma ou mais regras CSS.

Uma das ideas mais importantes do CSS é o conceito de especificidade, ou quão específicos devem ser os nossos selectores.

ESPECIFICIDADE É A CAPACIDADE DE UM SELECTOR DE SOBREPOR OUTROS NA SELECÇÃO DE UM OU MAIS ELEMENTOS PARA SOBRE ESTE APLICAR UMA OU MAIS REGRAS CSS

Especificidade

Imaginemos o seguinte cenário:

```
...
<head>
        <style>
                #main { background-color : green }
                .container { background-color : blue }
                div { background : red }
        </style>
</head>
<body>
        <div class="container" id="main">
                Olá eu sou um div
        </div>
</body>
```

Especificidade e a tabela dos pesos

Selector	Peso do Selector
Selector elemento	1
Selector classe	10
Selector id	100
Selector com atributo **style**	1000
!important	10000+

Especificidade

Isto significa que uma regra CSS sobre um elemento através do seu ID **tem mais peso** do que uma regra CSS sobre um elemento através da sua CLASSE. Ou seja selectores id são mais **específicos** que selectores classe.

Esta é uma outra diferença entre ids e classes - Os ids não somente devem estar associados a um só elemento, como também são mais específicos. Daí que a cor de fundo do nosso div no exemplo anterior deverá ser pintado a verde.

O efeito cascata só se aplica nos casos em que os selectores têm o mesmo grau de especificidade.

Especificidade

Modifiquemos um pouco o nosso exemplo, qual achas que será a cor do div

```
...
<head>
        <style>
                #main { background-color : green }
                .container { background-color : blue }
                div { background : red !important }
        </style>
</head>
<body>
        <div class="container" id="main" style="background:black">
                Olá eu sou um div
        </div>
</body>
```

Especificidade

Nós anteriormente mencionamos que usar estilização interna (através do atributo **style** em um elemento HTML) pode ter consquências nefastas - *especificidade forte é um destes efeitos controversos*. Estilização interna corresponde a 1000 pontos na tabela de pesos, muito maior do que a especificidade dos selectores ids ou classe.

O qualificador **!important** é o que maior peso tem. Tente o teu melhor para não usar estes métodos de estilização nos teus projectos.

No final desta aula, serás capaz de:

- Explicar o que é o Box Model
- Comparar e constrastar margin, padding e border
- Usar propriedades como **box-sizing** para melhor calcular a largura (width) e a altura (height) de um elemento.
- Explicar a diferença entre os elementos **block, inline-block** e **inline**
- Centralizar os elementos de forma vertical e horizontal usando **display**

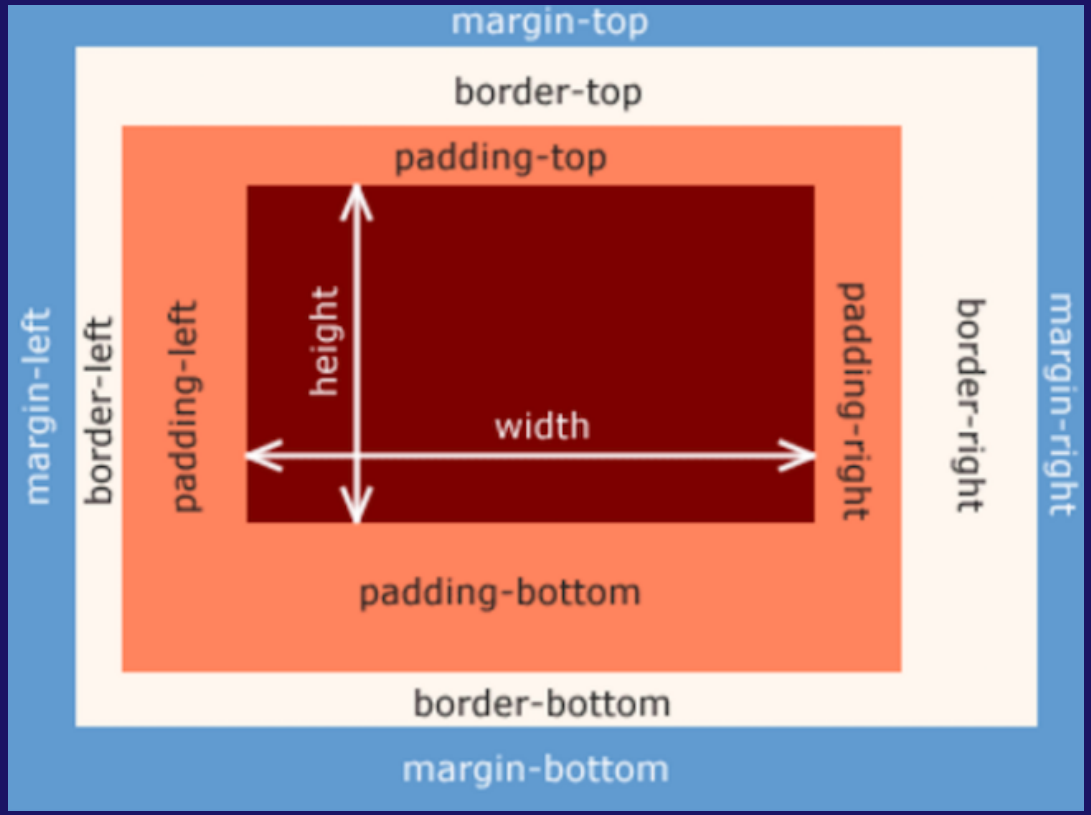
Box Model

Agora que temos um conhecimento elementar de css, vejamos como adiconar espaço entre os elementos e dentro dos elementos. Para fazer isto, nós precisamos antes entender o que o “box model” modelo caixa significa. Mas antes disto, vejamos algumas propriedades importantes para adicionar espaço entre e dentro dos elementos.

Box Model

- width - determina a largura da área de conteúdo de um elemento.
- height – determina a altura da área de conteúdo de um elemento.
- margin – Usado para gerar espaço a volta dos elementos. margin é um atalho para margin-top, margin-right, margin-bottom, margin-left.
- padding – Define a parte mais interna do modelo de caixa (Box Model), cria espaço entre a área do conteúdo de um elemento, dentro de qualquer margin (margem) ou border (borda) definida. padding é um atalho para padding-top, padding-right, padding-bottom e padding-left.
- border – A propriedade border define espaço entre o padding e margin. border é um atalho para border-top, border-right, border-bottom e border-left.

Box Model


margin-top
border-top
padding-top
margin-left
border-left
padding-left
height
width
padding-right
border-right
margin-right
padding-bottom
border-bottom
margin-bottom

Box Model

A verdadeira largura/altura de um elemento é composto do seu width/height + padding + border. **Margin não é contado quando calculamos a verdadeira altura/largura de um elemento.**

```
div {
        width : 200px;
        height : 200px;
        padding  : 20px;
        border : 20px solid black;
}
```

Verdadeira Largura = width (200px) + padding-left(20px) + padding-right(20px) + border-left(20px) + border-right(20px) = 280px

Box Model

A verdadeira largura/altura de um elemento é composto do seu width/height + padding + border. **Margin não é contado quando calculamos a verdadeira altura/largura de um elemento.**

```
div {
        width : 200px;
        height : 200px;
        padding  : 20px;
        border : 20px solid black;
}
```

Verdadeira altura = height (200px) + padding-top(20px) + padding-bottom(20px) + border-top(20px) + border-bottom(20px) = 280px

Propriedades adicionais CSS

Vejamos rapidamente duas propriedades CSS relacionadas ao modelo de caixa: box-sizing e border-radius.

```
div {
        box-sizing : border-box;
}
```

Por definição, a propriedade **box-sizing** é setada com o valor **content-box,** isto significa que o valor verdadeiro do **height** e **width** corresponde somente ao **width** e **height** da área de conteúdo. Quando configuramos a propriedade **box-sizing** com o valor **border-box**, os valores verdadeiros do **width** e do **height** passam a ser a soma do **padding**+**border**+**content**

Layout e a propriedade display

Agora que temos um conhecimento elementar do modelo caixa (ou Box Model), vejamos como podemos criar layouts usando a propriedade **display**.

Display *é a propriedade CSS mais importante para controlar o layout de uma página*. Todos os elementos têm um valor padrão para a propriedade **display** dependendo do tipo do elemento. O valor padrão para a maioria dos elementos é normalmente o valor block ou inline.

Layout e a propriedade display

Alguns valores que podem ser utilizados com a propriedade **display** são:

- **none** – Um elemento cuja propriedade **display** esteja configurado com o valor none não é renderizado (não aparece no ecrã).
- **block** – Um elemento block *começa em uma nova linha* e se extende para a esquerda tão longe quanto puder (até ocupar todo o espaço horizontal).
- **inline** – Um elemento inline pode ser colocado entre o texto de um parágrafo sem desfazer a estrutura do parágrafo. A tag <a>, e .
- **inline-block** – Um elemento inline-block é um valor híbrido entre o inline e o inline-block.

Inline-block vs inline vs block

Existe uma distinção importante entre os valores block e inline-elements que ainda não abordamos. Para entender vejamos um exemplo. O ficheiro deste exercício está no repositório https://github.com/LuisValdenecio/html_e_css_12inf na pasta display

Outros valores de display

Block, inline e inline-block são alguns dos valores mais usados com a propriedade **display**. No entanto, estes não são os únicos valores. Existe uma família de valores que nos permitem posicionar os nossos elementos em formato tabular. deste exercício está no repositório https://github.com/LuisValdenecio/html_e_css_12inf na pasta display.

OBS: Para melhorar a leitura do nosso HTML, é sempre conveniente não usar este método e optar sempre pela criação de tabela através da tag semântica <table>.

Alinhamento vertical

Uma possível excepcção a regra da observação anterior, tem que ver com o valor **table-cell**, que é utilizado para alinhar verticalmente um elemento no meio do seu elemento pai. Em geral, alinhamento vertical é relativamente complicado. Mas vejamos um exemplo de como este processo pode ser facilitado com o uso dos valores **table-cell** e **middle** nas propriedades **display** e **vertical-align.** Consulte o repositório abaixo na pasta display

https://github.com/LuisValdenecio/html_e_css_12inf na pasta display.

Alinhamento vertical

O valor table-cell obedece a uma outra propriedade (vertical-align) que permite o alinhamento vertical de um elemento.

No final desta aula, serás capaz de:

- Usar **float** para layouts simples
- Definir o que é o flow de um documento e como a propriedade **floating** altera o flow do documento
- Usar **clear** para trazer elementos de volta ao flow do documento
- Comparar e contrastar os valores **static**, **relative**, **asbolute**, e **fixed.**
- Alistar as características que a propriedade **position** confere a um elemento.
- Utilizar **position** para criar layouts mais complexos
- Entender o que é **flexbox** e como ele difere de outras formas de criação de layout
- Usar o **flexbox** para criar layouts sofisticados

Floats + clearing

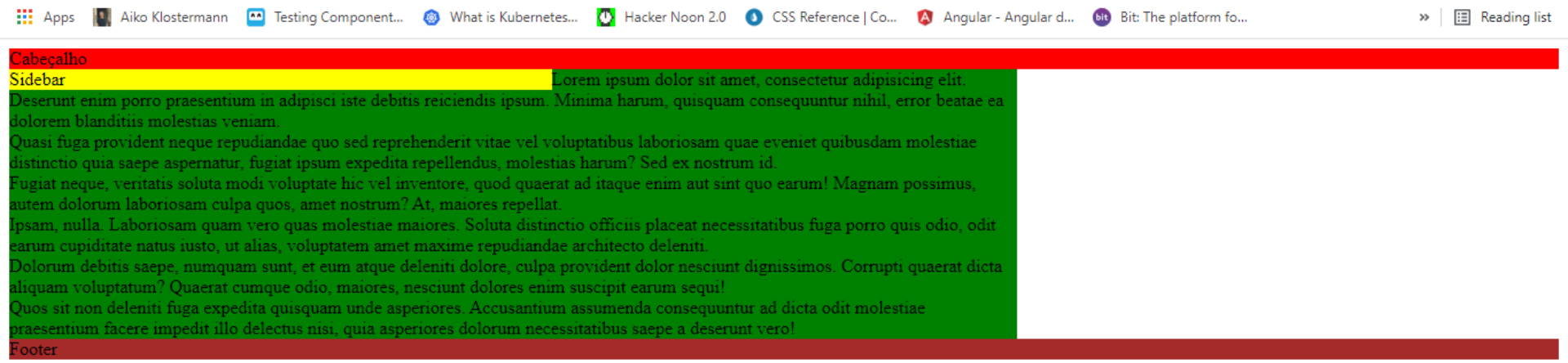
Uma outra forma de criar o layout de uma página é através da propriedade CSS **float.** Esta propriedade não é tão utilizada hoje, por conta de técnicas mais avançadas de criação de layouts. Mas, é importante conhecer os recursos que esta propriedade oferece.

Para entendermos a propriedade **float** precisamos entender o que significa o flow ou fluxo de um documento.

> *O Fluxo de um documento é o modelo pelo qual os elementos são renderizados por defeito no navegador. Neste modelo, os elementos são apresentados de acordo com o valor da propriedade **display**. Elementos **block** ocupam uma linha completa, elementos **inline** ocupam a mesma linha. Todos os elementos são organizados de cima para baixo.*

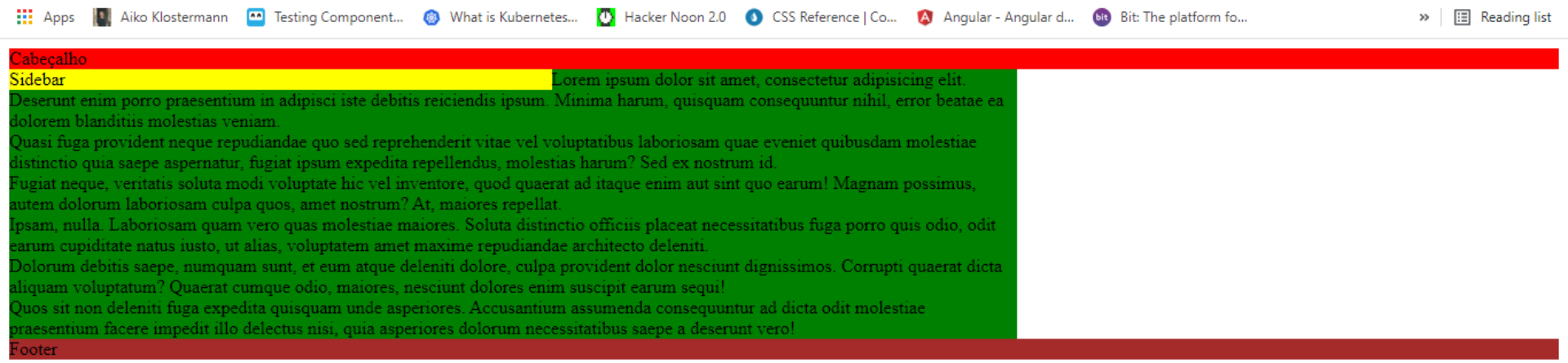
Floats + clearing

Quando aplicamos a propriedade **float** sobre um elemento, nós eliminamos este elemento do fluxo do documento. Consulte os ficheiros float.html e float.css no repositório https://github.com/LuisValdenecio/html_e_css_12inf abaixo da pasta layoutFloat. O float01.html é renderizado no navegador como na figura abaixo:



Floats + clearing

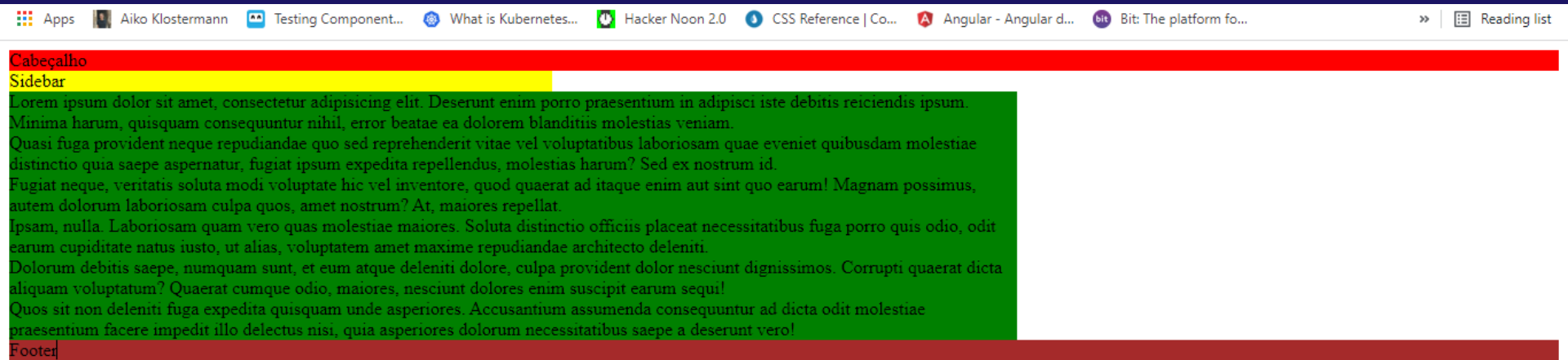
Como é possivel observar, o elemento **aside** está sobreposto ao elemento **article**. Isto acontece porque a propriedade **float** está configurada com o valor **left** e porque o elemento abaixo **article** não foi "cleared" ou limpo e como tal, **article** não voltou a disposição do fluxo do documento . Isto poderá ser o efeito que desejamos, se não for, precisamos utilizar a propriedade **clear** sobre **article**, de formas a que este volte ao fluxo do documento.



Floats + clearing

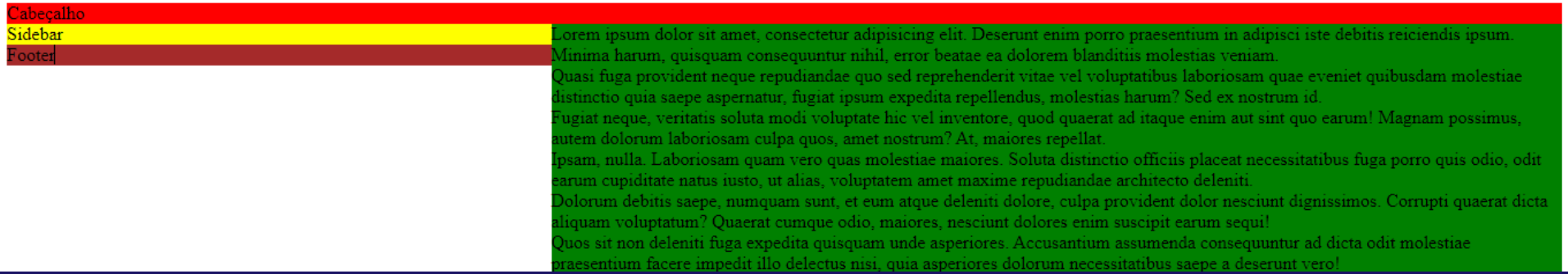
Para mudar este comportamento precisamos aplicar sobre o elemento **article** a seguinte regra CSS, **article** { clear : both }. O resultado final será como mostra a figura abaixo.

*OBS : A regra **clear : both** sobre **article** está comentada, elimine (apague os asteríticos e barras nos extremos da regra) o comentário sobre a regra para ver o efeito de clear : both em acção.*



Floats + overflow

Agora, vejamos como usar as propriedade **float** e **overflow** para melhorar um layout que tenha a seguinte aparência:



PROBLEMA : O elemento **aside** (barra amarela do lado esquerdo), tem uma altura menor que o elemento **article** (barra verde do lado direito) apesar de estarem na mesma linha. Para corrigir este problema, precisamos inserir algumas regras CSS (comentadas no ficheiro float01.css)

Floats + overflow

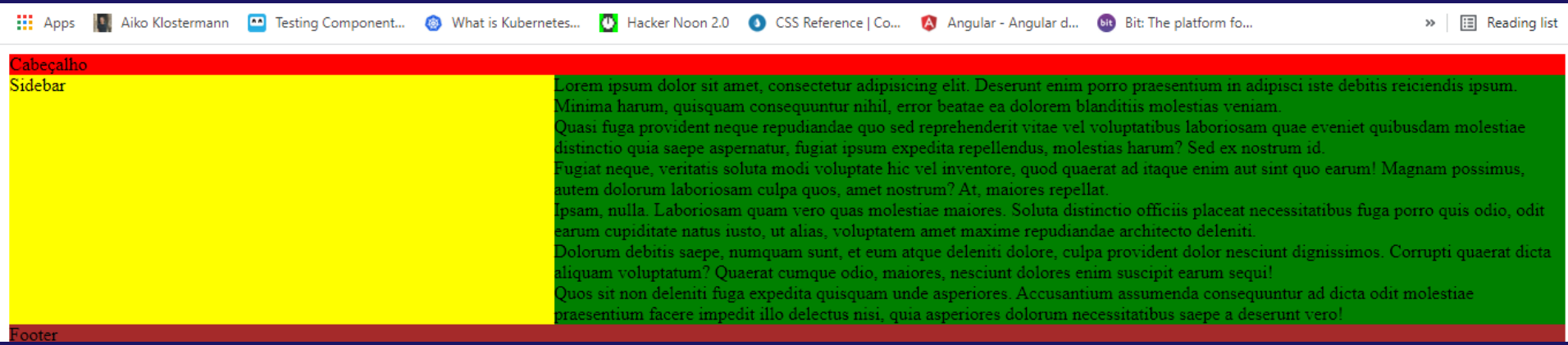
```
aside {
        float : left;
        padding-bottom : 1000px;
        margin-bottom : -1000px;
        background-color : green;
}

section {
        ...
        overflow : hidden;
        ...
}
```

Estas duas propriedades garantem que o elemento **aside** é grande o suficiente para ter a mesma altura que o elemento **article**

Elimina o espaço extra, permitindo que as duas caixas dentro de **section** tenham a mesma altura

Floats + overflow

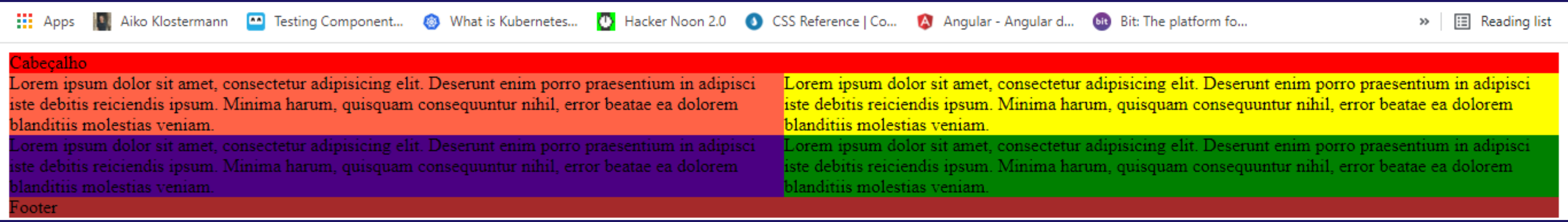
O resultado final será como nos mostra a figura abaixo:



A propriedade float

Se estas regras te parecem confusas, não te preocupes, havemos de ver formas mais simples de criar layouts como o flexbox nos próximos capitulos.

Treina as tuas habilidades com o float e cria um layout que se parece com este:



A propriedade position

Até a esta altura, nós vimos como usar as propriedades **display** e **float**, e vimos alguns exemplos de como usar estas propriedades para construír o layout de uma página. Nesta secção veremos uma outra importantíssima propriedade para os layouts : **position**.

A propriedade **position** nos permite controlar um elemento e determinar como e onde este pode ser posicionado. Esta posição é *relativa* a coisas como a posição inicial do elemento, outros elementos, ou mesmo a janela do navegador em si. Dependendo do tipo de posicionamento utilizado, o elemento poderá permanecer no fluxo do documento (posição relativa) ou ser removido do fluxo do documento (posição absoluta), exactamente como a propriedade **float.**

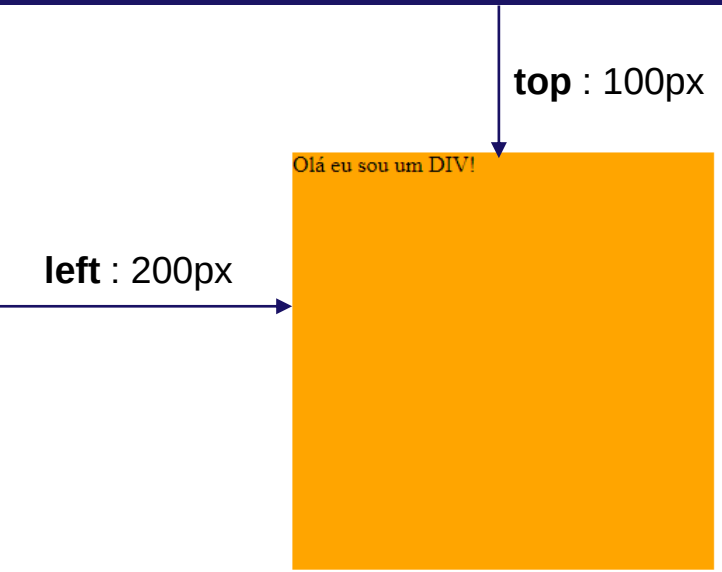
A propriedade position

Uma véz configurada a propriedade **position**, nós usamos propriedades adicionais como **right**, **left**, **bottom**, **top** e **z-index** (este último usado para sobrepor elementos).

Por definição, o valor da propriedade **position** é o valor **static**. Posição estática e relativa são basicamente a mesma coisa, com uma diferença importante : *Um elemento posicionado de forma **estática** não responde as propriedades **right, left, top, bottom** e **z-index**.*

Vejamos um exemplo disto:
OBS: O códifo fonte do exercício está no repositório https://github.com/LuisValdenecio/html_e_css_12inf, na pasta **position** nos ficheiros **position.html** e **position.css**

A propriedade position (relative)



O elemento está posicionado **100px** abaixo e **200px** a esquerda da sua posição padrão. O elemento **div** está com posição **relativa**, daí ser possível utilizar as propriedades **top** e **left**.

```
div {
    width: 300px;
    height: 300px;
    background: orange;
    position: relative;
    top: 100px;
    left: 200px;
  }
```

Ativar o Windows
Acesse Configurações para ativar o Windows.

A propriedade position (relative)

Quando a posição é estática (como no exemplo abaixo, por termos comentado a regra **position : relative**), o elemento não obedece as propriedades **top, left, right** ou **bottom.**

```
div {
    width: 300px;
    height: 300px;
    background: orange;
    /*position: relative;*/
    top: 100px;
    left: 200px;
  }
```

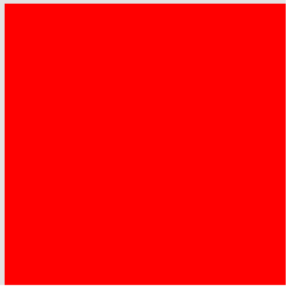
Ativar o Windows
Acesse Configurações para ativar o Windows.

Posição relativa vs posição absoluta

Quando os elementos têm posição **relativa,** eles *não são removidos do fluxo do documento*, e os valores das propriedades **top, bottom, left** e **right** vão posicionar o elemento relativo a sua posição padrão (com base no fluxo do documento). No exemplo anterior, a propriedade **top** está setada com o valor 100px; isto significa que o **div** vai estar 100 pixels abaixo de onde deveria estar.

Quando os elementos têm posição **absoluta**, a situação é diferente. Nesta situação, *o elemento é removido do fluxo do documento*, e qualquer valor para as propriedades **top, bottom, right** e **left** será relativa ao seu elemento pai. Desde que o elemento pai esteja estaticamente posicionado, se o elemento pai estiver estaticamente posicionado, o elemento será posicionado relativo a um ancestral *com posicionamento não estático.*

Posição relativa vs posição absoluta



```
body {
    margin: 0; /*REMOVE AS MARGENS DO NAVEGADOR*/
  }

.red {
    background-color: red;
    position:  absolute;
    width: 200px;
    height: 200px;
    bottom: 0;
    right: 0;
}

.wrapper {
    width: 500px;
    height: 400px;
    background: #e3e3e3;
    position: relative;
    margin-bottom: 10px;
}
```

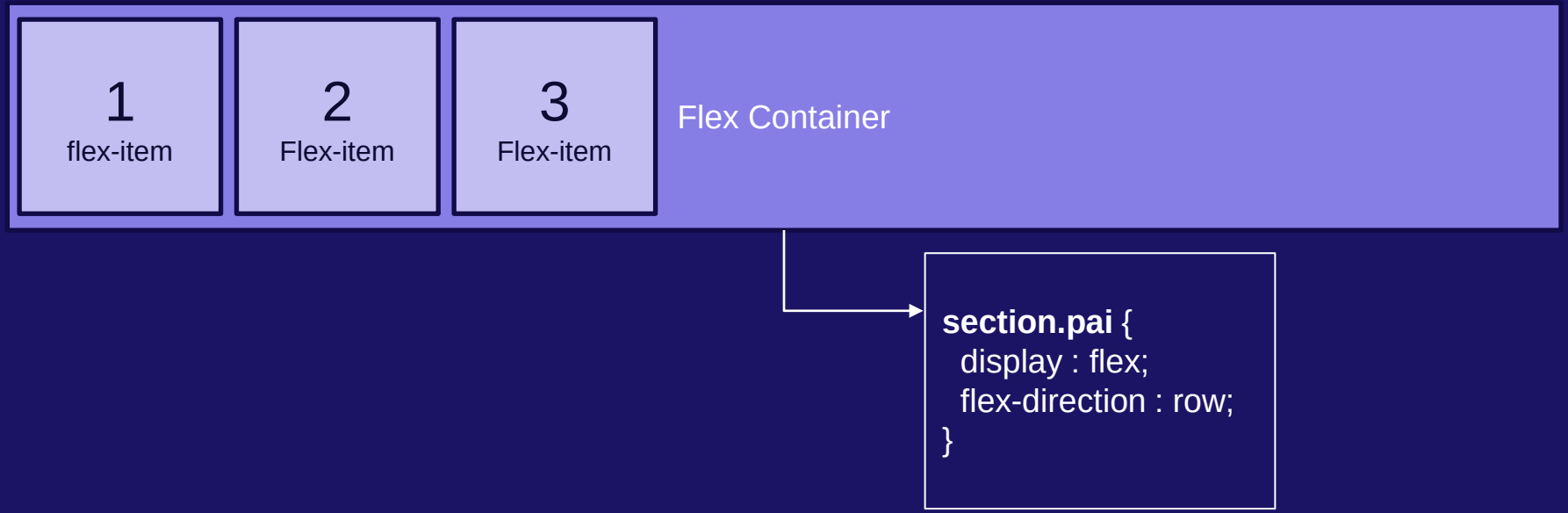
Estas duas propriedades e os respectivos valores garantem que o div (vermelho) é posicionado no canto inferior direito do elemento pai (div cinzento)

Ativar o Windows
Acesse Configurações para ativar o Windows.

Posição fixa

O quarto tipo de posicionamento é o **fixed (fixo)**. Este posicionamento é parecido ao posicionamento absoluto. Os elementos com posicionamento absoluto estão SEMPRE posicionados relativos ao viewport actual. Isto significa que eles não movem nem quando utilizamos o scroll para navegar a parte de baixo da nossa página. Para ver um exemplo, consulte os ficheiros fixedPosition.html e fixedPosition.css no repositório https://github.com/LuisValdenecio/html_e_css_12inf na pasta position. Note o uso da propriedade z-index no elemento **div#front.**

Flexbox

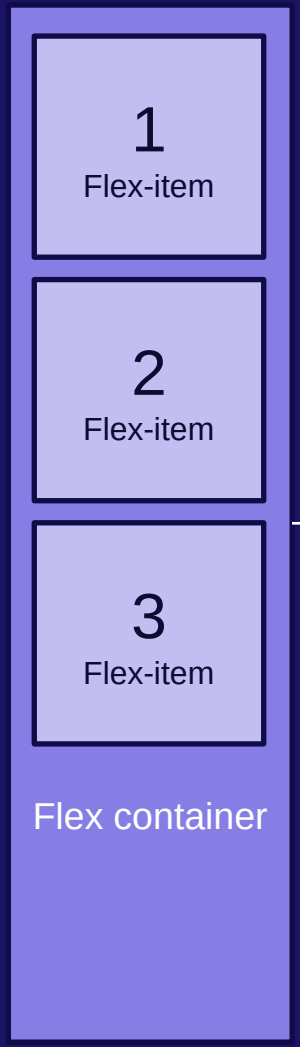

1
flex-item
2
Flex-item
3
Flex-item
Flex Container
section.pai {
display : flex;
flex-direction : row;
}

Flexbox

Flex container

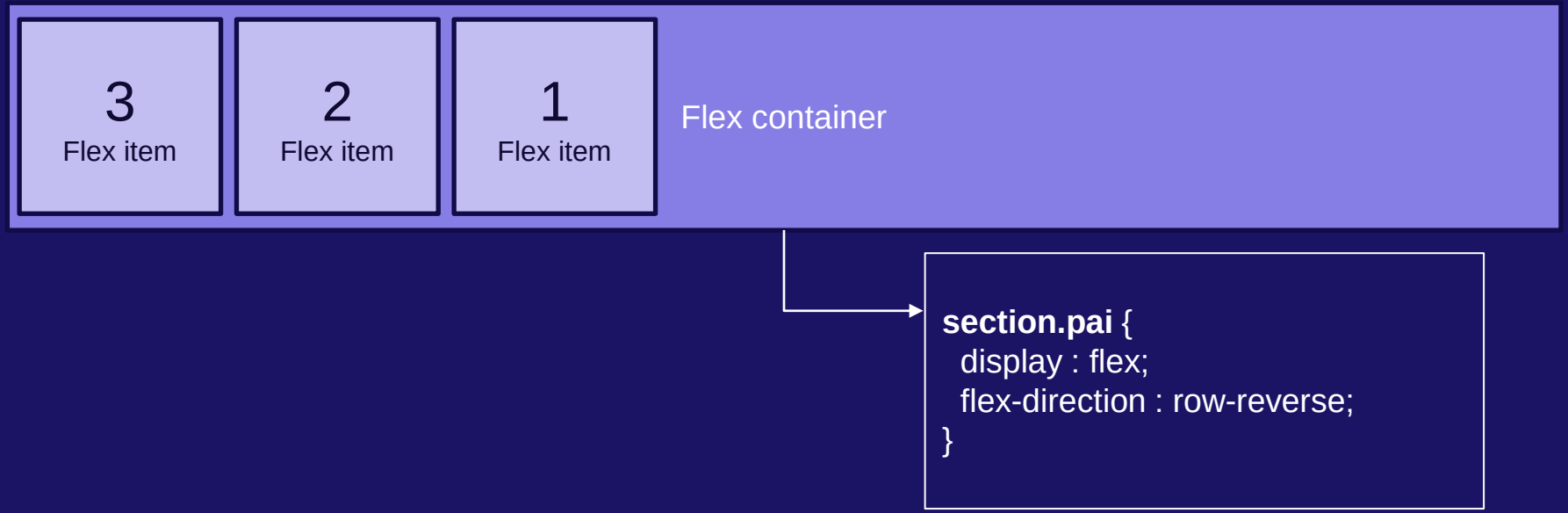
```
section.pai {
  display : inline-flex;
  flex-direction : row;
}
```

Flexbox

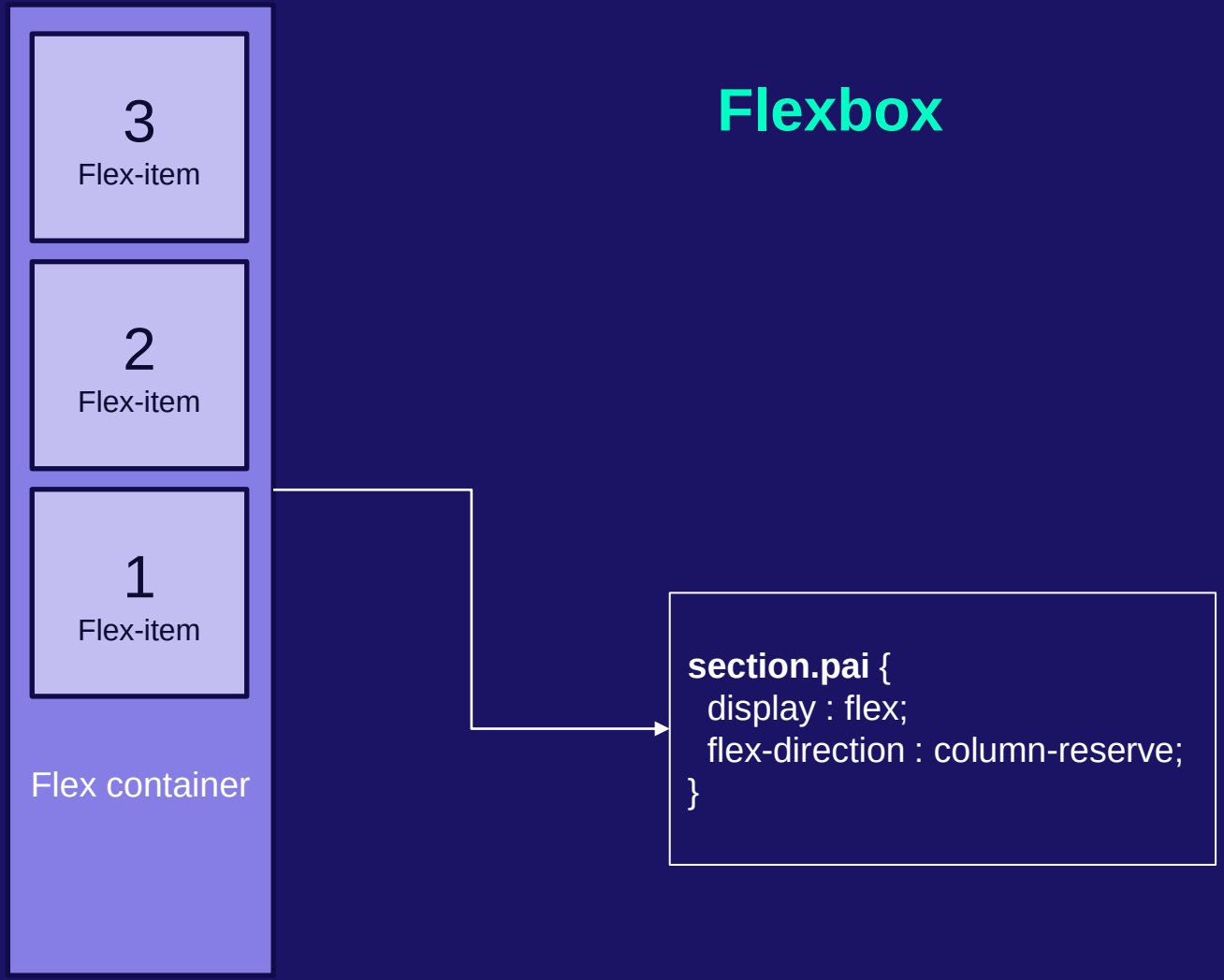



```
section.pai {
  display : flex;
  flex-direction : column;
}
```

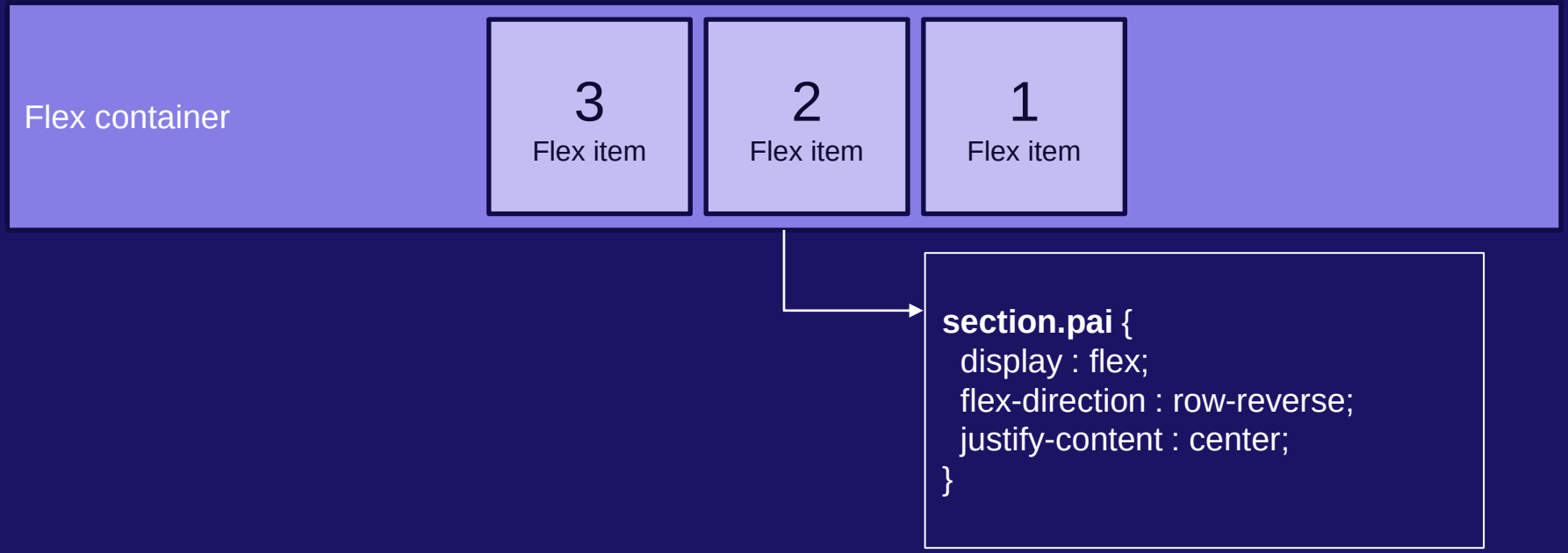
Flexbox


3
Flex item
2
Flex item
1
Flex item
Flex container
section.pai {
display : flex;
flex-direction : row-reverse;
}

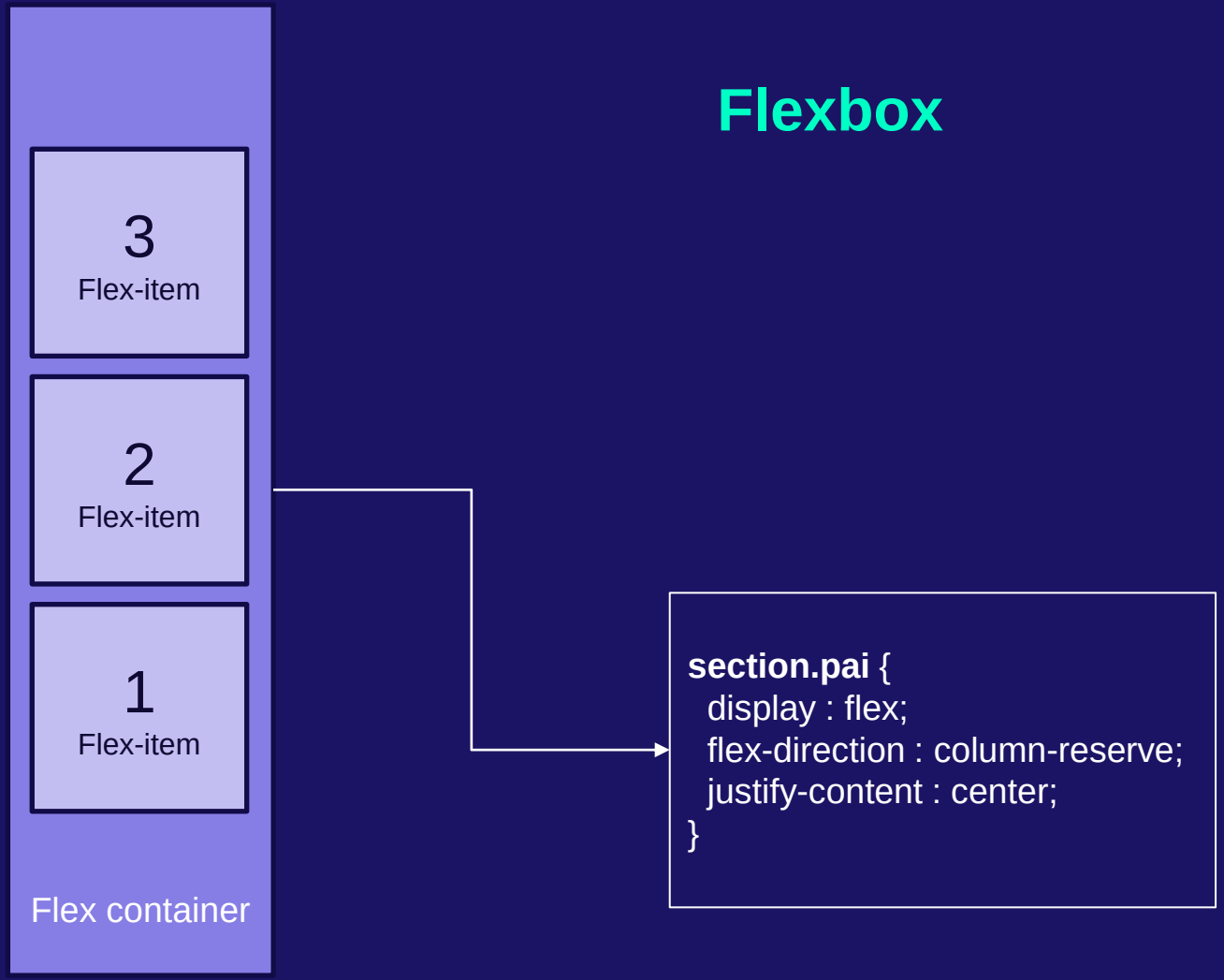
Flexbox


3
Flex-item
2
Flex-item
1
Flex-item
Flex container
section.pai {
display : flex;
flex-direction : column-reserve;
}

Flexbox


Flex container
3
Flex item
2
Flex item
1
Flex item
section.pai {
display : flex;
flex-direction : row-reverse;
justify-content : center;
}

Flexbox


3
Flex-item
2
Flex-item
1
Flex-item
Flex container
section.pai {
display : flex;
flex-direction : column-reserve;
justify-content : center;
}

Flexbox


```
section.pai {
  display : flex;
  flex-direction : row-reverse;
  justify-content : center;
  align-items : center;
}
```

Flexbox

Consulte o repositório https://github.com/LuisValdenecio/html_e_css_12inf na pasta flexbox, nos ficheiros flexbox01.html e flexbox01.css. Aqui vai uma breve consideração das propriedades do CSS, utilizadas com o flexbox.

- display - No flexbox é imperativo usar esta propriedade no elemento pai. A propriedade display no flexbox pode ser configurada com dois valores : flex e inline-flex (um elemento flex ocupa a linha toda, um elemento inline-flex, ocupa somente o que for necessário para apresentar o se conteúdo.
- flex-direction - Esta propriedade determina a orientação dos flex-items dentro de um flex container (horizontal - row ou vertical - column)

Flexbox

- align-content - Nós podemos pensar nesta propriedade como a que nos permite alinhar os flex-items de forma vertical. Os possíveis valores para esta propriedade são:
  - flex-start (limite superior do flex container)
  - center (Centro vertical do flex container)
  - flex-end (limite inferior do flex-container)

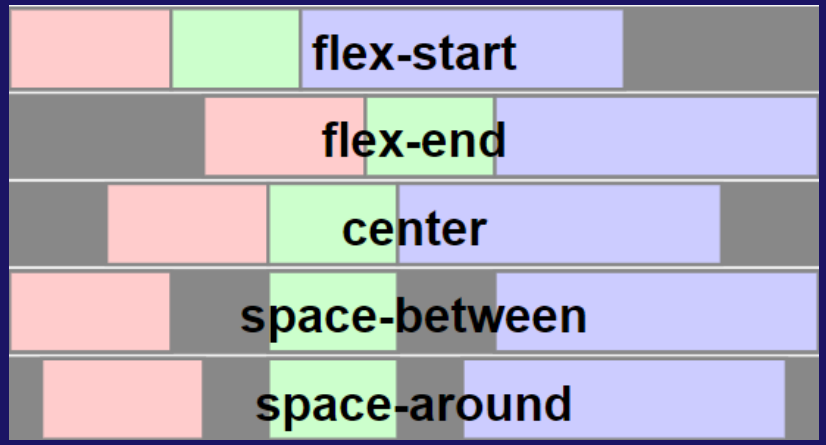
Note que quando o valor de **flex-direction** for **column** ou **column-reserve**, o efeito da propriedade **align-cotent** para a ser horizontal

Flexbox

- Justify content - Determina a distribuição dos espaço extra entre os **flex-items.** Os possíveis valores para esta propriedade são:
  - flex-start (Posiciona os flex-items do lado esquerdo)
  - center (Centraliza os flex-items no flex container)
  - flex-end (Posiciona os flex-items do lado diretio)
  - space-between (Posiciona os flex-items de modos que haja o mesmo espaço *entre* cada um deles)
  - Space-around (Posiciona os flex-items de modos que haja o mesmo espaço a volta de cada um deles).

Flexbox

A figura baixo ilustra o efeito de cada uma das propriedades do slide anterior.



Note que quando o valor de **flex-direction** for **column** ou **column-reserve**, o efeito da propriedade **justify-content** para a ser vertical.

Mais propriedades Flexbox

- **order** – Esta propriedade nos permite ordenar elementos baseados na posição de cada um deles no nosso documento HTML. **(Quanto mais alto o valor de order, mais abaixo o elemento é posicionado).**

- **flex-basis** – Permite que mudar o tamanho da área de conteúdo. Atenção que flex-basis não é o mesmo que width. Consulte este post no StackOverflow para saber mais.

- **flex-grow –** Permite setar o tamanho de um flex-item em proporção do tamanho do flex container. Por exemplo, se um elemento tem o valor 2 para a propriedade flex-grow e outro 1 para a mesma propriedade, significa que o primeiro elemento vai ocupar duas vezes mais espaço que o segundo.

Mais propriedades Flexbox

- **flex-shrink –** Esta propriedade é parecida com a anterior, mas serve para diminuir o tamanho dos elementos quando não há espaço suficiente. Consulte este link para ver o efeito desta propriedade.

- **flex** – é uma propriedade atalho para flex-basis, flex-grow e flex-shrink. Permite configurar o valor destas três propriedades em uma única linha.

- **flex-wrap** – Quando um flex-container não grande o suficiente para acomodar informação, esta propriedade determina se o flex container pode ou não ser enlargado.

No final desta aula, serás capaz de:

- Explicar o que a propriedade **transform** faz
- Alistar os métodos que podem ser valores da propriedade **transform**
- Usar a propriedade **transform** para manipular o layout de uma página
- Como estruturar o valor da propriedade **transform** no CSS
- Usar a propriedade **transition** para animações simples sem JavaScript
- Explicar o que é uma keyFrame
- Usar a propriedade **animation** junto com as regras @keyframe para criar animações CSS mais complexas
- Aprender a criar um relógio analógico usando somente HTML e CSS

CSS Transforms

A Propriedade **transform** é uma tecnologia nova do CSS que *nos permite mover elementos na página* por (translate), (rotate), (scale). Algumas vezes, esta propriedade é usada de forma isolada, noutras ocasiões, esta propriedade pode ser encontrada em conjunto com as animações CSS (que veremos adiante nesta secção.

Só existe uma propriedade que precisamos saber para usar **transform : transform.** No entanto, esta propriedade pode ser configurada com um vasto leque de valores. Vejamos alguns destes valores (no próximo slide)

CSS Transforms

transform - Esta propriedade CSS altera o tamanho e a posição do conteúdo, e nos permite mover (translate), girar (rotate), multi ou mini-dimensionar (scale), e torcer (skew) elementos.

Os métodos que efectivamente permitem as operações são:

- **translate** - O método translate() move um elemento da sua actual posição. Por exemplo, a regra transform : translate(50px, 100px); move um elemento 50px a direita e 100px para baixo.

- **rotate** - O método rotate() gira um elemento no sentido horário ou anti-horário de acordo ao grau fornecido. Por exemplo, a regra transform : rotate(90deg); ,o elemento vai rotates 90º no sentido horário.

CSS Transforms

- **scale** – O método scale() aumenta ou diminui o tamanho de um elemento (de acordo aos parámetros fornecidos as propriedades width e height). Por exemplo, se fornecermos a um elemento a regra transform : scale(2,3);, o elemento deverá aumentar a sua dimentsão 200% na horizontal e 300% no vertical.

Vejamos o exemplo, na pasta transform do repositório https://github.com/LuisValdenecio/html_e_css_12inf.

CSS Transition

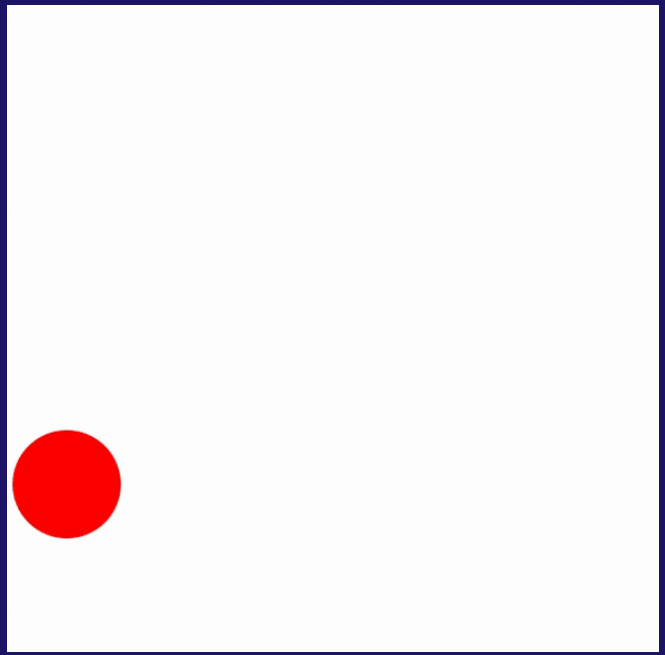
Tal como **transform**, **transition** é uma tecnologia nova do CSS. Esta propriedade nos permite animar os elementos da nossa página sem precisarmos recorrer ao JavaScript. O JavaScript é óptimo para animações complexas e detalhadas, para animações simples, o CSS pode atingir os mesmos resultados.

Para usar transitions ou transicções no CSS nós precisamos fazer uso da propriedade **transition.** Esta propriedade nos permite setar e controlar a velocidade das animações quando alteramos as propriedades CSS.

Em vés de fazer com que as alterações nos valores das propriedades tomem efeito imediato, nós podemos fazer com que as alterações das propriedades ocorram com o tempo.

CSS Animation

Nós vimos como criar animações usando a propriedade **transition**, e se quiséssemos ter animações mais detalhadas como na figura abaixo?



CSS Animation

Existe uma outra propriedade CSS, também utilizada para criar animações, esta propriedade é a **animation**, que é utilizada para definir animações usando somente CSS. Esta propriedade torna possível transicções complexas de uma propriedade CSS para outra.

CSS Animation

Para criar uma animação, usando CSS animation, nós precisamos:

1. Configurar a propriedade **animation** no elemento HTML que deverá ser animado, o tempo de duração de todos os keyframes e se a animação deve ou não ser infinita.

2. Definir um conjunto de keyframes ou etapas, que descrevam o processo da animação. Por exemplo, a animação que precisamos construír, é composta de três partes:

a) Inicialmente, o nosso elemento div é um círculo vermelho.
b) Depois, o nosso elemento div é um quadrado azul
c) No final da animação, o nosso elemento é um quadrado verde

CSS Animation

Quando estivermos a criar uma animação usando keyframes no CSS, nós precisamos nomear a nossa animação, antes de poder usar. O nome da animação deve ser apropriado ao trabalho que a animação executa.

Vejamos o exercício na pasta **animations** no repositório https://github.com/LuisValdenecio/html_e_css_12inf.

Exercício de animações e transformações

No final desta aula, serás capaz de:

- Explicar o que é o CSS Reset e porque são usados
- Explicar o que é a filosofia “Responsive and mobile design first”
- Explicar o que são media queries
- Saber escrever media queries para criar desenhos responsivos
- Listar diferentes unidades de medida para desenho responsivo
- Comparar e constrastar as unidades **em** e **rem.**
- Entender boas práticas para escrever HTML e CSS
- Explicar o que é o Twitter Bootstrap

CSS Reset

No desenvolvimento de um site, é sempre ideal que a aparência do projecto seja universal em todos os navegadores. No entanto, isto não ocorre por definição. Por exemplo, o Google chrome tem uma formatação padrão de 8px de margem para todas as páginas - este efeito pode não ser o desejado.

Existe um recurso, conhecido como CSS Reset que é basicamente um ficheiro com a extensão .css, que nos ajuda com a padronização da aparência do nosso site - O CSS Reset faz com que o nosso site se pareça de igual forma em todos os navegadores.

Existem muitos CSS Resets. O que mais nos interessa para este curso é o normalize.css, que tem um excelente suporte para sites responsivos.

Desenho Responsivo

Um site é responsivo quando é capaz de se adaptar ao “viewport” ou tamaho da tela do utilizador.

Desenho Responsivo

Para o desenvolvedor, a filosofia “Responsive + Mobile First Design” é a forma de desenhar sites considerando inicialmente a boa experiência de uso dos dispositivos móveis e posteriormente utilizadores desktop.

Como os utilizadores não têm todos o mesmo tamanho de tela, o desenho responsivo leva em conta este detalhe. E lida com isto de forma adequada, proporcionado para cada dispositivo a melhor forma de apresentar a informação.

De forma prática, a filosofia “Responsive + Mobile First Design” significa escrever primeiro CSS para dispositivos moveis e somente depois, escrever CSS para o restante dos dispositivos.

Desenho Responsivo

Para começar a criar desenhos responsivos, nós precisamos primeiro:

1. Aprender a usar media queries para escrever CSS baseado no tamano de tela
2. Usar unidades de medidas relativas e não absolutas.

Media Queries

Um aspecto fundamental do desenho responsivo é a possibilidade de ter diferentes estilos para diferentes telas. A forma como conseguimos este efeito é atravéz dos media queries:

```
@media screen and (max-width : 480px) {
        /*todo o estilo declado aqui será somente para dispositivos móveis*/
}
```

Ver os exercício mediaqueries01.html e mediaqueries02.html na pasta mediaQueries do repositório https://github.com/LuisValdenecio/html_e_css_12inf.

Unidades de medidas responsivas

Até a esta altura nós vimos duas unidades de medida, pixels (px) e percentagens (%). O Pixel é uma unidade absoluta de medida. Unidades absolutas são excelentes quando temos ideas bem definidas sobre o tamanho de um determinado elemento. Pixels, centímetros, milimetros e pontos são unidades de medidas absolutas.

As unidades relativas, tal como sugere o nome são relativas ao tamanho de um outro elemento. As unidades de medida do CSS são a percentagem, ems, root ems, exes, etc.

Unidades de medidas responsivas

- % - Define o tamanho de um elemento como uma percentagem do tamanho total do elemento que o contém.
- em – 1em – É igual ao tamanho padrão da fonte do elemento, isto nos permite configurar o tamanho de propriedades relativo ao tamanho da fonte.
- rem : Parecido a unidade anterior, mas relativo ao elemento root (html).
- vh - 1vh : É igual a 1/100 da altura do viewport (tela do navegador)
- vw – 1vw : É igual a 1/100 da largura do viewport (tela do navegador)

Dicas para escrever código HTML e CSS manutenível

- Para projectos pequenos, não use muitos Ids e classes, em véz disto, use selectores mais complexos (descedente, adjacente, enésimo filho, etc.)

- Mantenha o CSS objectivo, não repita as mesmas regras entre vários selectores.

- Use comentários CSS para dividir diferentes partes da formatação. Por exemplo, formatação do cabeçalho deve ter uma secção própria, formatação da secção principal uma outra parte e do mesmo geito com outras secções do site.

Exercício de desenho responsivo

Para este exercício, vamos construír um layout responsivo. A nossa tarefa é criar as seguintes classes:

- `col-4` – consome 33% de espaço, quando o tamanho da tela for maior que 960 pixels, 50% de espaço, quando o tamanho da tela for maior que 765 pixels e 100% quando o tamanho da tela for inferior a 765 pixels.
- `col-6` – consome 50% de espaço, quando o tamanho da tela for maior que 960 pixels e 100% quando o tamanho da tela for menor que 960 pixels.
- `col-12` – consome 100% do espaço para todos os tamanhos de tela.

Twitter Bootstrap

O Twitter Bootstrap é um framework construído e mantido gratuítamente pelo Twitter. É uma colecção de ferramentas que nos ajudam a escrever sites responsivos com maior facilidade e agilidade.

Para usar o Twitter Bootstrap nós temos a disposição três alternativas:

1. Descarregar o ficheiro que contém o código bootstrap
2. Usar uma CDN (Content Delivery Network) para acessar ao mesmo ficheiro sem necessariamente ter uma cópia na nossa máquina
3. Instalar através de um gestor de pacotes e dependências como o bower ou o npm.

Twitter Bootstrap

A forma mais simples é atravéz de um CDN. Se formos para o endereço : https://getbootstrap.com/docs/4.0/getting-started/download/#bootstrap-cdn, nós podemos copiar e colar a tag link no nosso Html, passamos imediatamente a ter acesso as funcionalidades do bootstrap.

Os atributos adicionais na tag link são necessários para garantir segurança sempre que se deseja extraír algum conteúdo de um CDN.