

Módulo 3 - CSS

De maneira objetiva o CSS é uma maneira de dar estilo ao código criado pelo HTML, a linguagem funciona como uma camada de personalização do conteúdo visível. Imagine a seguinte situação, você que já conhece as tags HTML, decide criar um botão verde para o seu site. Utilizando HTML, você estrutura o seu site criando o botão - `<button>`, e para colorir o seu botão de verde você utilizaria o CSS! Além disso, com essa ferramenta é possível posicionar os elementos HTML, fazer animações e tornar o layout funcional para diversos tipos de dispositivos.

Apesar de simples, o CSS acaba sendo uma linguagem fundamental, e muito poderosa! Veja só no exemplo a seguir a diferença entre um site que possui somente **HTML** e um site estruturado com **HTML + CSS**.

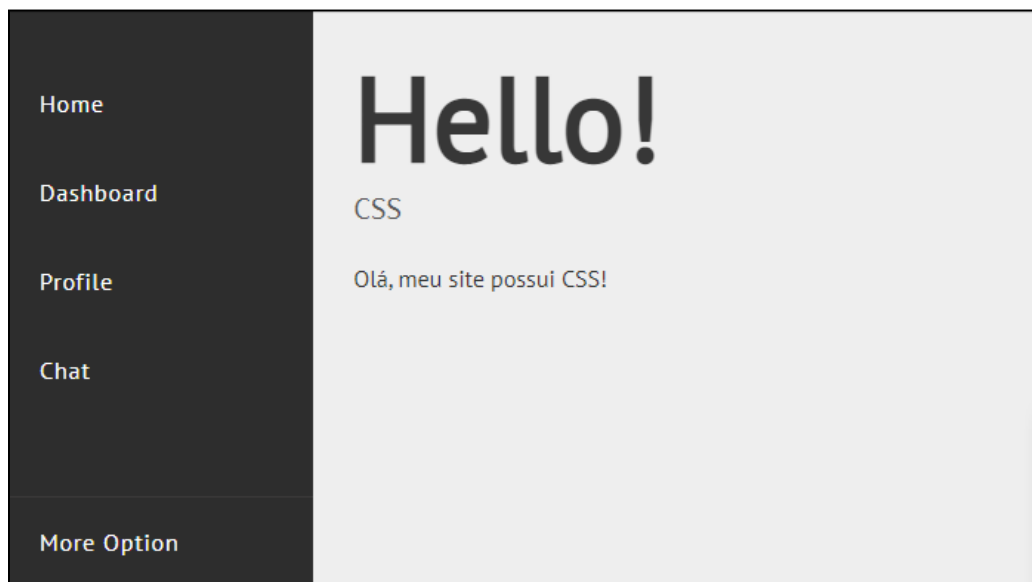


Imagem 59 - Menu desenvolvido com CSS e HTML



Imagem 60 - O mesmo site sem HTML.

Repare que o mesmo site na segunda imagem, não possui nenhum tipo de estrutura (bloco), cores ou fontes personalizadas! Com HTML nós fazemos apenas o esqueleto do nosso site, e com o CSS criamos o design.

3.1 Sintaxe básica CSS

A sintaxe básica de estrutura de códigos em CSS é bem simples! O código é composto por um seletor (*tag*, *id* ou *class*) e duas chaves, onde as propriedades css serão escritas, veja o exemplo:

```
Seletor {  
    propriedade: valor;  
    propriedade: valor;  
    propriedade: valor;  
    propriedade: valor;  
}
```

Imagine a seguinte situação: nós queremos criar um bloco retangular cinza para estruturarmos um menu. Para isso, deve ser utilizado CSS, certo? Vamos lá, imagine um bloco da seguinte maneira:



Imagem 61 - Bloco cinza para um menu.

Utilizando a sintaxe que vimos anteriormente, a lógica para escrever esse código seria mais ou menos essa:

```
caixa-menu{  
    altura: 10cm;  
    largura: 30cm;  
    cor-de-fundo: cinza;  
}
```

caixa-menu seria o seletor, ou seja, poderia ser uma tag, classe ou id (Nos aprofundaremos um pouco mais sobre esses conceitos no decorrer deste módulo) altura, largura e cor-de-fundo seriam as propriedades e, acompanhado das propriedades estão os valores.

Agora, fique com um exemplo real, no qual as propriedades e valores estão escritas em inglês, conforme a **documentação** da linguagem. Nesse exemplo, foi criado um título <h1> no HTML, que foi alinhado ao centro da tela, e teve sua cor alterada para verde.

Código HTML:

```
<body>

    <h1>
        Título do site
    </h1>

</body>
```

Código CSS:

```
h1{
    text-align: center;
    color: green;
}
```

Resultado:



Conforme foi dito anteriormente, esse exemplo foi criado a partir da documentação da linguagem, isso quer dizer que: através do CSS existe apenas uma maneira de se mudar a cor de uma fonte, que é a propriedade **color**. As propriedades CSS são determinadas pela

documentação, ou seja, se olharmos esse documento, encontramos a propriedade color e a sua utilização, que é para a mudança de cor das fontes. As propriedades do CSS foram todas baseadas na língua inglesa, que é um idioma universal. Clique [aqui](#) para ver a documentação do CSS.

3.2 Maneiras de aplicar o CSS no projeto

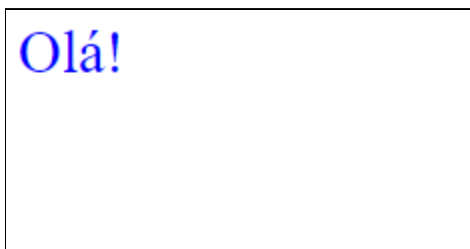
De maneira objetiva, é possível aplicar CSS à estrutura HTML de três maneiras: inline, interno e externo.

CSS inline

Como o nome sugere, utilizando o CSS inline, nós iremos escrever os códigos CSS através de um **atributo** “style”, **dentro** das tags html. Por exemplo, se quiséssemos mudar o tamanho da fonte e a cor de um parágrafo utilizando o atributo style, seria da seguinte forma:

```
<p style=" font-size: 30px;">  
  Olá!  
</p>
```

Resultado:



Vale ressaltar que essa maneira de se utilizar o CSS não é muito recomendada, pois os códigos CSS e HTML estão sendo escritos em uma mesma linha, isso dificulta na organização do código e também na identificação de erros.

CSS interno

Com o CSS interno, nós também escrevemos os códigos dentro do arquivo html, porém, os códigos são escritos dentro de uma **tag** <style>. Vejamos um exemplo: vamos mudar o estilo de fonte e o espaçamento de um texto utilizando o CSS interno.

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML CSS JS</title>
  </head>
<body>

  <style>
    p{
      font-family: sans-serif;
      letter-spacing: 10px ;

    }
  </style>

  <p>
    Texto
  </p>

</body>
</html>
```

Resultado:

T e x t o

CSS externo

Como o nome diz, o CSS externo será desenvolvido em um arquivo externo ao HTML, ou seja, utilizamos um arquivo diferente com a extensão “.css”. Essa é a maneira mais recomendada de se aplicar CSS às páginas, visto que, nesse caso cada arquivo irá exercer a sua determinada função, deixando os códigos e arquivos mais organizados, facilitando a identificação de erros e tornando o desenvolvimento mais rápido. Além disso, podemos reutilizar essas mesmas folhas de estilo para estilizar outras páginas.

Para que seja possível a utilização do CSS externo, é necessário que seja determinado em nosso documento HTML uma ligação entre ele e a folha de estilo. Essa indicação é feita dentro da tag <head>, através da tag <link> de um documento HTML. Dessa maneira:

```
<!DOCTYPE html>
<html lang="pt-br">

  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="style.css">
    <title> Meu documento </title>
  </head>

  <body>
```

```
<p>
    Texto verde!
</p>


</body>
</html>
```

Repare que dentro da tag link, foram utilizados dois atributos para especificar o documento: rel e href. O atributo “rel” especifica a relação do CSS com o documento HTML, que é de folha de estilo e o atributo “href” determina o arquivo que será utilizado para estilizar a página, nesse caso, o arquivo “style.css”.

Seguindo adiante, para mudarmos a cor do parágrafo acima, o código escrito dentro do arquivo “style.css” deve ser da seguinte maneira:

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

Resultado:



Texto verde!

3.2 Estilização de textos

Tipografia

Com css, podemos alterar fontes da mesma maneira que alteramos as cores. A propriedade responsável por essa alteração é a “font-family”. A propriedade “font-family” pode receber seu valor com ou sem aspas, dependendo de sua formação, quando o nome da fonte tiver espaço, por exemplo.

Existem basicamente, dois tipos de fonte: serifadas (serif) e não serifadas (sans-serif). As fontes serifadas possuem um pequeno ornamento em suas terminações, já as não serifadas não. Veja o exemplo abaixo na **Imagem XX**.



Imagem 62 - Comparação fontes

Fonte - Medium, Julio Queiroz

Além disso, o exemplo acima também mostra um exemplo de fonte de cada “família”, ou seja, a fonte “Times New Roman” é uma fonte serifada, já o Arial é sem serifa. É possível também, verificar se há a disponibilidade de fonte no navegador no qual o documento foi aberto, caso não, o CSS irá selecionar outra fonte que for especificado como fonte principal da página! Veja um exemplo:

```
p{  
    font-family: "Times New Roman", "Times", serif.  
}
```

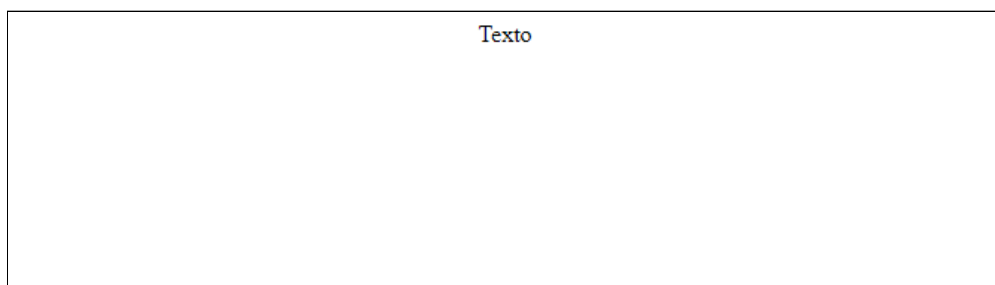
Nesse caso, se o navegador não tiver a fonte “Times New Roman”, ele irá exibir apenas a fonte “Times”, e, no último caso, o navegador apenas exibirá a fonte principal serifada. Outra propriedade bastante utilizada é a “font-style”, que teremos como valor o estilo de fonte que queremos, exemplo: itálico (italic), normal e oblíqua (oblique).

Alinhamento

Agora que já conhecemos algumas propriedades CSS, e entendemos melhor como a sua sintaxe funciona, vamos aprender um pouco mais sobre a configuração dos textos. Uma propriedade bastante utilizada é o `text-align`, que nos permite alinhar o texto no centro do elemento pai, por exemplo.

```
p{  
    text-align: center;  
}
```

Resultado:



Existem outros valores para essa propriedade, como por exemplo `right` e `justify`, que servem para alinhar o texto à direita e justificar o texto, respectivamente. Nós podemos ver outros valores para as propriedades através da documentação.

Veja abaixo outras propriedades importantes e as suas definições:

```
p{
    letter-spacing: 2px; /* tamanho do espaçamento entre cada letra */
    word-spacing: 3px; /* tamanho do espaçamento entre cada palavra */
    Line-height: 2px; /* tamanho da altura de cada linha */
}
```

3.5 Cores

Existem três padrões para a especificação de cores no CSS, que são: nome, padrão RGB(a) e valor hexadecimal. Por exemplo, podemos colorir um texto para vermelho de três formas, veja abaixo:

```
p{
    color: rgb(255, 000, 000); /* RGB */
    color: red; /* NOME */
    color: #FF0000; /*HEXADECIMAL */
}
```

A primeira representação é o RGB, a abreviação representa as três cores primárias, que são: vermelho (red), verde (green) e azul (blue). Com o arranjo dessas três cores, nós podemos criar qualquer cor do espectro visível que quisermos! Um exemplo, se juntarmos as cores red e green, formamos o amarelo. Nesse padrão, o amarelo seria representado da seguinte forma:

rgb (255, 255, 000)

Ou seja, a cor vermelha possui o valor “cheio” que é 255, e o verde também, já o azul não possui nenhuma unidade. Para variar a tonalidade das cores, basta utilizar os valores que estão entre 0 e 255 na representação. A segunda representação é o **nome**, essa representação é baseada no nome da cor em inglês.

A última representação é o hexadecimal. Essa representação é uma das mais recomendadas, pois ela possui uma representação universal nos navegadores e também é a forma

mais curta e objetiva de se representar uma cor. Esse padrão é baseado no RGB, temos 6 caracteres, os dois primeiros indicam o canal vermelho, os dois seguintes o verde, e os últimos o azul.

Não é necessário que decoremos todos os padrões de cores e seus valores, uma maneira mais fácil de trabalhar com cores é a utilização de ferramentas como o Adobe Color, que nos fornece um círculo cromático com todos os padrões de cores disponíveis. Para acessar a ferramenta, clique [aqui](#).

Utilizamos esses padrões como valores para todas as propriedades CSS que possuem a configuração “color”, como por exemplo: `background-color` (Para definir a cor de **fundo** dos elementos), `color` (Para definir a cor dos **textos**) e `border-color` (Para definir a cor das bordas) .

3.4 Imagens de fundo

As **imagens de fundo** devem ser manipuladas por CSS, através da propriedade `background-image`. Veja um exemplo abaixo, da manipulação de uma imagem no fundo do **corpo** de uma página:

```
body{

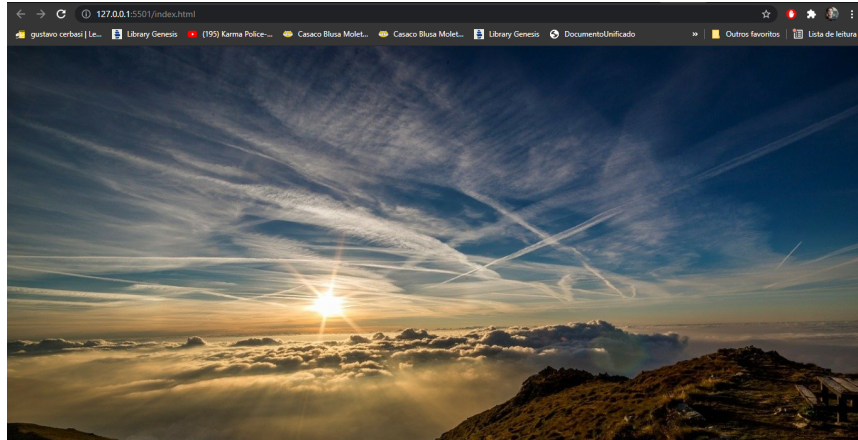
    background-image: url(..img/imagem-de-fundo.jpg); /* selecionando a
imagem */

    background-position: center; /* a imagem foi posicionada no centro do
elemento */

    background-size: cover; /* nesse exemplo, a imagem deve cobrir todo o
fundo do elemento, independente do tamanho da tela */

}
```

Resultado:



Vale ressaltar que, as imagens ficam no **fundo** dos elementos, ou seja, se colocarmos outros elementos dentro de `<body>`, eles irão aparecer por cima da imagem.

3.4 Espaçamento e sizing

Existem algumas maneiras de se trabalhar com espaçamento e dimensões de elementos em CSS. Imagine que você queira aumentar a distância entre um texto e uma imagem, por exemplo, qual propriedade CSS você usaria? E, em qual elemento você aplicaria essa propriedade? Para isso, vamos compreender alguns conceitos importantes!

Box-model

O motor de renderização do navegador representa os elementos como uma "caixa", seguindo o padrão definido pelo CSS, conhecido como "box-model". Sendo assim, o conteúdo do elemento é uma das quatro partes que compõem a caixa, sendo as demais o seu preenchimento (padding), borda (border) e margem (margin). A Imagem XX nos dá uma representação desse modelo.

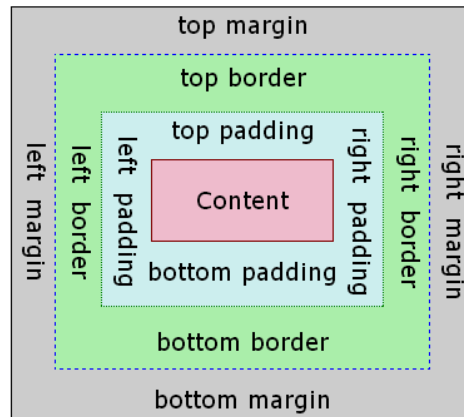


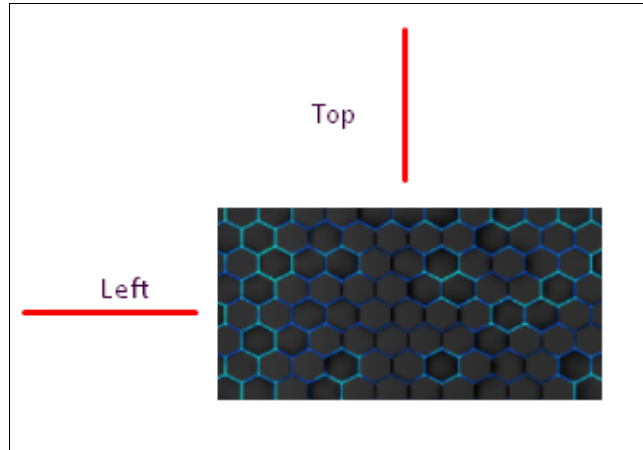
Imagem 63 - Box-model

Margin

Nós podemos, por exemplo, definir a altura e largura de uma imagem (content) através das propriedades “width” e “height”. Ou definir uma margem para esse elemento, através da propriedade “margin”, e escolher o sentido para qual a margem será aplicada, veja o exemplo:

```
img{  
    width: 200px; /*largura da imagem*/  
    height: 100px; /* altura da imagem */  
    margin-top: 100px; /* margem no topo */  
    margin-left: 100px; /* margin à esquerda */  
}
```

Resultado:



Border

Podemos também, colocar uma borda em um elemento através da propriedade **border**, veja o exemplo:

```
p {  
    font-family: sans-serif;  
    color: #CE5937;  
    border: 1px black solid; /* definição das propriedades da borda  
    */  
}
```

Resultado:

There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable. If you are going to use a passage of Lorem Ipsum, you need to be sure there isn't anything embarrassing hidden in the middle of text. All the Lorem Ipsum generators on the Internet tend to repeat predefined chunks as necessary, making this the first true generator on the Internet. It uses a dictionary of over 200 Latin words, combined with a handful of model sentence structures, to generate Lorem Ipsum which looks reasonable. The generated Lorem Ipsum is therefore always free from repetition, injected humour, or non-characteristic words etc.

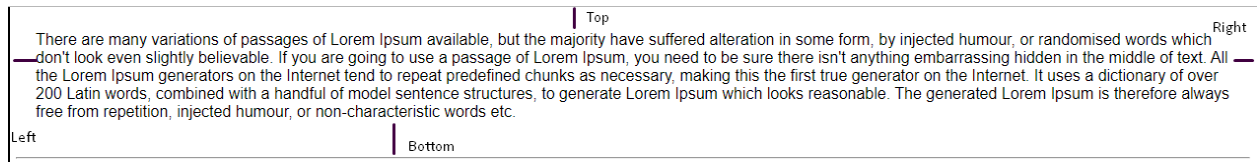
Veja que a tag `<p>` agora possui uma borda com 1px de grossura de cor preta e sólida. Se quiséssemos aplicar essa borda apenas no topo, por exemplo, deveríamos utilizar então, a propriedade “border-top”.

Padding

Por último, temos a propriedade **padding**, que serve para definir o espaçamento interno dos elementos (a distância entre o conteúdo e a borda). Vamos então, colocar um espaçamento interno em todas as direções de uma vez dentro de um parágrafo:

```
p{  
    padding: 20px;  
}
```

Resultado:



Quando colocamos a propriedade “padding” direto, e não especificamos a direção, o valor é aplicado para todos os sentidos, conforme mostra o exemplo. Essa situação também é pertinente para outros atributos como margin e border. Porém, da mesma forma, poderíamos especificar apenas um padding-top por exemplo, apenas para a parte de cima do parágrafo.

Box-sizing

Todo elemento no HTML renderizado como uma caixa. Controlamos seu tamanho com width, sua borda com border e ainda temos as margens externas e espaçamentos internos com o padding. O **box-model** dita como todas essas propriedades se relacionam para determinar o tamanho final do elemento.

Com a propriedade **box-sizing**, podemos trocar essa renderização padrão do **box-model** para outra. Um valor bastante utilizado nessa propriedade é o **border-box**. Com essa configuração, os valores de border e padding são incorporados mantendo a largura inicial definida do elemento, facilitando o posicionamento e o cálculo dos elementos. Veja um exemplo prático:

HTML

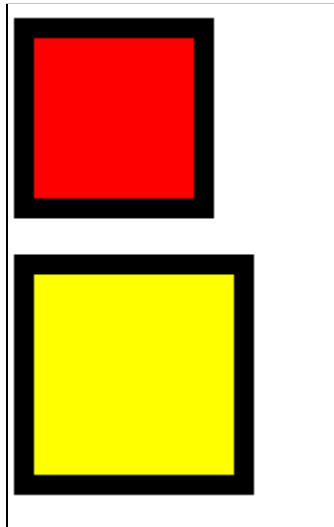
```
<div id="elemento-um">  
  
</div>  
  
<div id="elemento-dois">  
  
</div>
```

CSS

```
#elemento-um{  
    box-sizing: border-box;  
    height: 100px;  
    width: 100px;  
    background-color: red;  
    border: black solid 10px;  
  
}  
  
#elemento-dois{  
  
    height: 100px;  
    width: 100px;  
    background-color: yellow;  
    border: black solid 10px;
```

}

Resultado:



Reparem que, apesar de termos adicionado uma borda de dez pixels no elemento vermelho, a propriedade `box-sizing` manteve a largura do elemento, que é de 100px. O valor da borda foi incorporado. Já no segundo exemplo, isso não acontece. O valor da borda é acrescentado à largura (`width`).

3.5 Divs

De maneira objetiva, a tag `<div>` define um bloco em uma página de documento HTML. Diante disso, essa tag é muito utilizada quando precisamos agrupar elementos sem a necessidade do uso de um **elemento semântico**³. Isso acontece pois o elemento `div` não possui um **valor semântico**. Sendo assim, não representa nenhum significado para o navegador ou para os mecanismos de buscas.

Por ser bastante utilizado para agrupar elementos, seu uso acaba sendo voltado para organizar informações dentro de um layout. Dessa forma, é possível formatar e manipular os elementos, inclusive a própria **<div>**, através do CSS de uma forma organizada. Geralmente é acompanhada de atributos de ID e classe, para poder facilitar essa organização e formatação.

³Um elemento semântico serve para especificar o tipo de conteúdo que será inserido nas tags, deixando o código mais organizado, facilitando o serviço dos navegadores e indexadores de conteúdo. Exemplo de tag semântica: `<figure>`.

Então, se quiséssemos construir uma caixa para colocar um parágrafo dentro, poderíamos construir essa estrutura utilizando uma div, veja só:

HTML:

```
<body>
```

```
    <div>
```

```
        <p>
```

```
        There are many variations of passages of Lorem Ipsum
        available, but the majority have suffered alteration in
        some form, by injected humour, or randomised words which
        don't look even slightly believable. If you are going to
        use a passage of Lorem Ipsum, you need to be sure there
        isn't anything embarrassing hidden in the middle of text.
        All the Lorem Ipsum generators on the Internet tend to
        repeat predefined chunks as necessary, making this the
        first true generator on the Internet. It uses a dictionary
        of over 200 Latin words, combined with a handful of model
        sentence structures, to generate Lorem Ipsum which looks
        reasonable. The generated Lorem Ipsum is therefore always
        free from repetition, injected humour, or
        non-characteristic words etc.
```

```
        </p>
```

```
        </div>
</body>
```

CSS:

```
p {
    font-family: sans-serif;
    padding: 20px;
    text-align: justify;
    color: white;
}
```

```
div{
    height: 300px;
    width: 500px;
    background-color: green;
    padding: 20px;
}
```

Resultado:

There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable. If you are going to use a passage of Lorem Ipsum, you need to be sure there isn't anything embarrassing hidden in the middle of text. All the Lorem Ipsum generators on the Internet tend to repeat predefined chunks as necessary, making this the first true generator on the Internet. It uses a dictionary of over 200 Latin words, combined with a handful of model sentence structures, to generate Lorem Ipsum which looks reasonable. The generated Lorem Ipsum is therefore always free from repetition, injected humour, or non-characteristic words etc.

3.6 Classes e ids

As **classes** são uma forma de identificar um grupo de elementos. Utilizando classes, podemos aplicar uma **mesma** configuração para vários elementos de uma só vez. Veja um exemplo:

CSS:

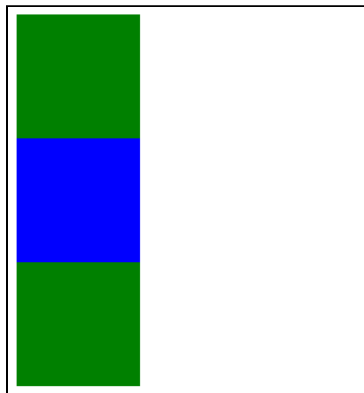
```
.caixa-verde{ /* as classes são representadas por um "." no CSS */  
  
    height: 100px;  
    width: 100px;  
    background-color: green;  
  
}
```

```
.caixa-azul{ /* as classes são representadas por um "." no CSS */  
  
    height: 100px;  
    width: 100px;  
    background-color: blue;  
  
}
```

HTML:

```
<body>  
    <div class="caixa-verde"> </div>  
    <div class="caixa-azul"> </div>  
    <div class="caixa-verde"> </div>  
  
</body>
```

Resultado:



Repare que uma mesma configuração feita por CSS, está sendo aplicada em duas tags, através do atributo “class”, cujo nome é “caixa-verde”. A configuração não foi colocada diretamente na tag **<div>**, porque se fizéssemos isso, a configuração também seria aplicada na **<div>** de cor azul.

Já os **ids** são uma forma de identificar um elemento, e devem ser **únicas** para cada elemento. É uma identificação única para o elemento, como se fosse um RG. Por meio delas,

pode-se atribuir formatação a um elemento em especial. Os ids são representados através de “#”, veja o exemplo:

CSS:

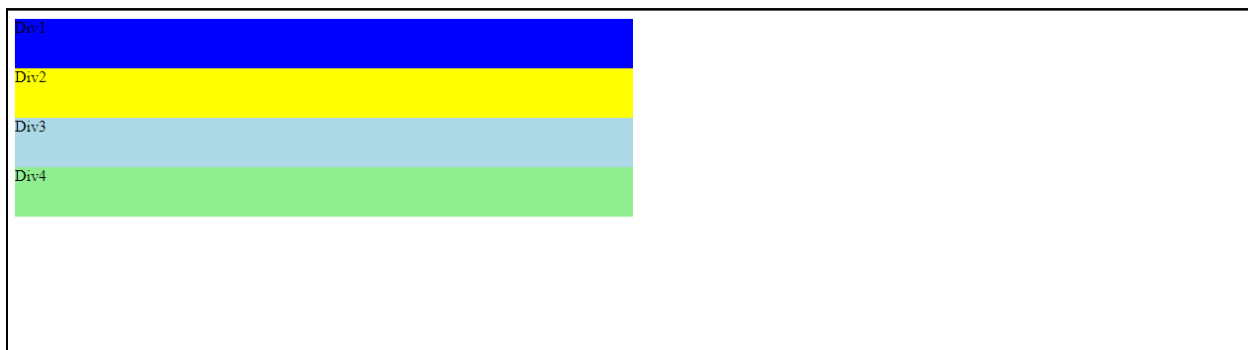
```
#primeira {  
    background-color: blue;  
    height: 50px;  
    width: 50%;  
  
}  
#segunda {  
    background-color: yellow;  
    height: 50px;  
    width: 50%;  
}  
#terceira {  
    background-color: lightblue;  
    height: 50px;  
    width: 50%;  
}  
#quarta{  
    background-color: lightgreen;  
    height: 50px;  
    width: 50%;  
}
```

HTML:

```
<body>  
    <div id="idUm">Div1</div>  
    <div id="idDois">Div2</div>  
    <div id="idTres">Div3</div>  
    <div id="idQuatro">Div4</div>
```

</body>

Resultado:



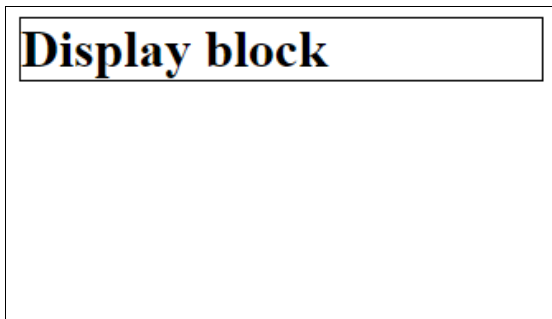
3.7 Display e position

3.7.1 Display

O display é uma das propriedades mais importantes para o CSS, visto que, ela é uma das responsáveis por definir o tipo de layout que será exibido dos elementos. Cada elemento possui um valor padrão para a sua exibição (display). Existem 4 tipos importantes de configuração de display, são eles: Block, Inline, None e Inline-block.

Block

Com essa definição, o elemento se comporta como um bloco, ocupando toda a largura disponível na página. Veja alguns exemplos que possuem esse padrão já definido: **<p>**, **<h1>**, **<div>**. Exemplo:



Inline

A propriedade inline faz com que o elemento HTML seja renderizado como uma linha, ocupando a menor quantidade de espaço possível. Veja o exemplo:



Nome

Essa configuração faz com que o bloco não seja renderizado, ou seja, o elemento não é exibido quando aplicamos a definição display: none.

Inline-block

Essa definição faz com que o elemento seja renderizado como um bloco (ou seja, podemos definir altura e largura), e uma linha (que ocupa um espaço mínimo de largura).

3.7.2 Position

Existem sete tipos de valores para a propriedade `position`, são eles: `static`, `absolute`, `fixed`, `relative`.

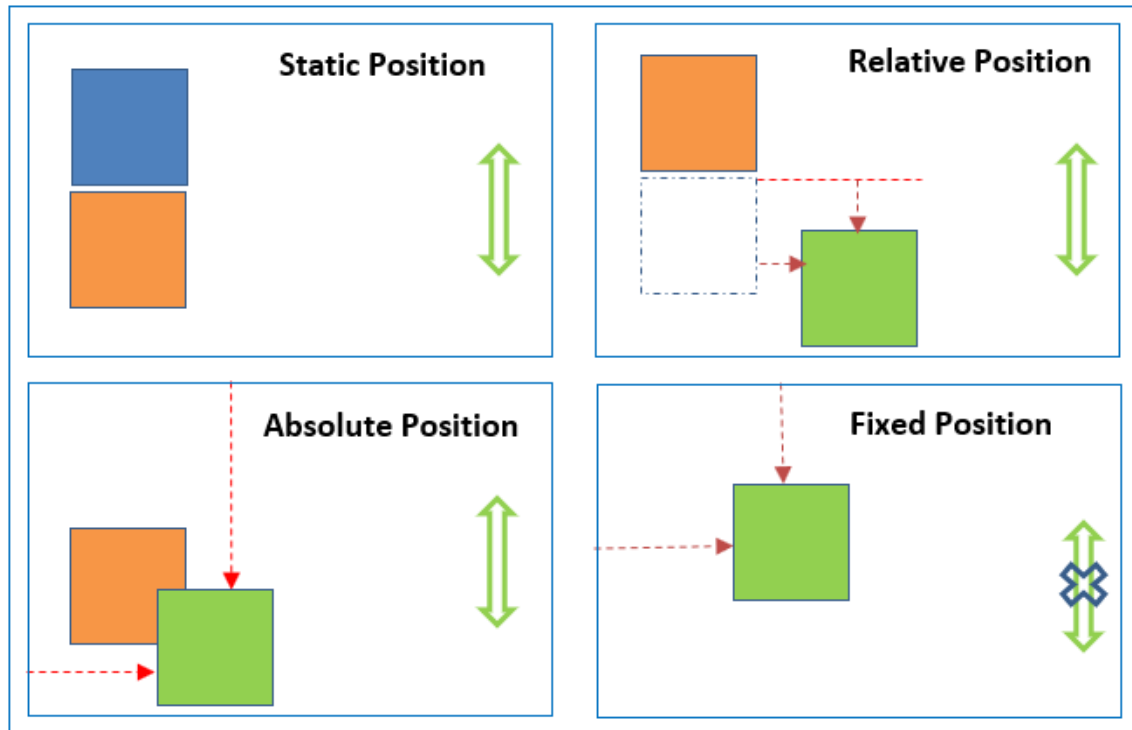


Imagem XX - Principais valores para `position`.

Fonte:

Static

Essa é a definição padrão para os elementos, eles são renderizados em ordem, da maneira que são especificados no decorrer do documento.

Absolute

O elemento é posicionado em relação ao seu elemento pai que tenha uma posição diferente de `static`.

CSS:

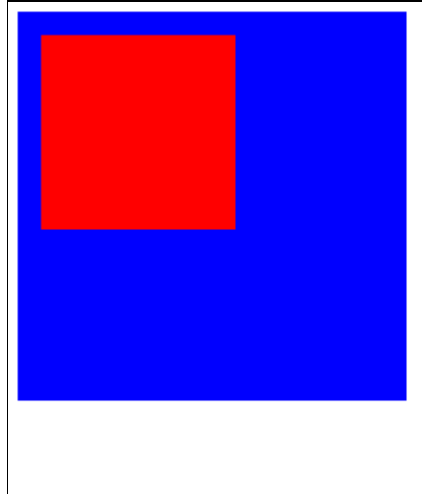
```
.pai{  
  height: 200px;  
  width: 200px;  
  background-color: blue;  
  
}
```

```
.filho {  
  position:absolute;  
  width:100px;  
  height:100px;  
  background:red;  
  left: 20px;  
  top: 20px;  
}
```

HTML:

```
<body>  
  
  <div class="pai">  
    <div class="filho">  
  
      </div>  
  
    </div>  
  
</body>
```

Resultado:



Fixed

Essa configuração faz com que o elemento seja posicionado em relação à janela de visualização do usuário. Então, se rolarmos a página, por exemplo, o elemento ficará fixado na janela e não no fluxo padrão do documento HTML.

Relative

O `position relative` posiciona o elemento em relação a si mesmo. Ou seja, o ponto zero será o canto superior esquerdo, e ele começará a contar a partir dali.

3.8 Bordas

Através da propriedade `border` podemos definir bordas para os elementos HTML. Veja um exemplo:

HTML:

```
<body>
```

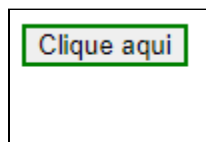
```
<button>
  Clique aqui
</button>

</body>
```

CSS;

```
button{
  border: 2px green solid;
}
```

Resultado:



Criamos uma borda verde, sólida, com um pixel de “grossura”. Podemos definir outros tipos de borda: pontilhada (dotted) e duplicada (double). Isso vale para todos os elementos que são renderizados como blocos ou inline-blocos: imagens, caixas de input, botões, divs, parágrafos ou títulos.

Existe também uma propriedade responsável por arredondar a borda dos elementos, que é o `border-radius`, veja um exemplo:

HTML:

```
<body>

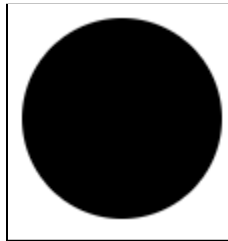
<div class="box-um">
```

```
</div>  
</body>
```

CSS:

```
.box-um{  
  height: 100px;  
  width: 100px;  
  background-color: black;  
  border-radius: 50px;  
}
```

Resultado:



Criamos uma div quadrada, e arredondamos as suas bordas, dando forma à um círculo perfeito.

Responsividade com CSS

O seletor `@media` é utilizado para aplicarmos as configurações de responsividade para diferentes layouts. Cada configuração será aplicada em função do tipo de tela que está sendo reproduzida. Mais especificamente a sua largura (width).

Os tipos de mídia podem ser:

- all (O tipo padrão. Usado para todos os tipos de dispositivos e layout.)

- print (Para impressão)
- screen (Para todos os tipos de tela: telefones, tablets, computadores)
- speech (Para leitura automática de páginas)

Veja uma breve explicação sobre a sintaxe dessa funcionalidade:

```
@media tipo de mídia (largura de exibição do dispositivo){
```

Código css desejado para a condição

```
}
```

Agora, vejamos um exemplo mais prático:

```
@media screen and (max-width: 600px) {  
  div.example {  
    display: none;  
  }  
}
```

Nesse código, configurado para uma screen (tela), irá ocultar uma div quando a largura do navegador tiver 600px ou menos.

Além disso, podemos criar diferentes arquivos CSS para diferentes tipos de tela, dessa maneira:

```
<link rel="stylesheet" media="screen and (min-width: 900px)"  
href="widescreen.css"> <!-- Telas grandes -->
```

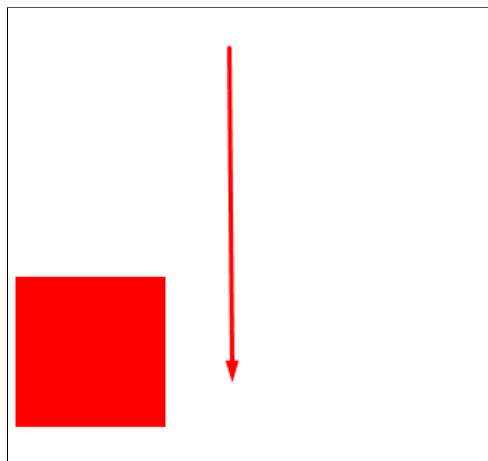
```
<link rel="stylesheet" media="screen and (max-width: 600px)"  
href="smallscreen.css"> <!-- Telas pequenas -->
```

Animações

O seletor `@keyframes` é o responsável por criar animações através do CSS. O uso dessa funcionalidade é bem simples e intuitiva. Uma animação é criada, mudando gradualmente de um conjunto de estilos para outro. No período da animação, podemos alterar o conjunto de estilo várias vezes. Veja um exemplo:

```
div {  
  width: 100px;  
  height: 100px;  
  background: red;  
  position: relative;  
  animation: mymove 5s infinite;  
}  
  
@keyframes mymove {  
  from {top: 0px;}  
  to {top: 200px;}  
}
```

Resultado:



Nesse exemplo, a div que possui a identificação de “mymove” se move gradualmente 200px de cima para baixo. A animação possui cinco segundos e irá se repetir infinitamente,

como está descrito nas propriedades da div. Veja um pouco mais as possibilidades desse recurso aqui, na [documentação](#).