Capítulo 5

ESTRUTURA DE UM DOCUMENTO HTML

Um documento HTML válido precisa seguir obrigatoriamente a estrutura composta pelas tags `<html>`, `<head>` e `<body>` e a instrução `<!DOCTYPE>`. Esta estrutura está informada em uma documentação que descreve todos os detalhes do HTML, no caso as tags e atributos, e como os navegadores devem considerar e interpretar estas tags, esta documentação é chamada de "especificação do HTML", e através do que está declarado nela que é possível entender se um documento HTML válido. Um documento HTML inválido é carregado pelo navegador, porém em um "*modo de compatibilidade*", vamos entender melhor sobre isto logo mais.

Abaixo, vamos conhecer em detalhes cada uma das tags estruturais obrigatórias:

5.1 A TAG <HTML>

Na estrutura do nosso documento, antes de começar a colocar o conteúdo, inserimos uma tag `<html>`. Dentro dessa tag, é necessário declarar outras duas tags: `<head>` e `<body>`. Essas duas tags são "irmãs", pois estão no mesmo nível hierárquico em relação à sua tag "mãe", que é `<html>`.

```
<html> <!-- mãe -->
  <head></head> <!-- filha -->
  <body></body> <!-- filha -->
</html>
```

5.2 A TAG <HEAD>

A tag `<head>` contém informações sobre o documento HTML que são de interesse somente do navegador e para outros serviços da web, e não para as pessoas que vão acessar nosso site. São informações que não serão exibidas diretamente no navegador, também podemos considerar um local onde informamos os metadados sobre a página.

A especificação do HTML obriga a presença da tag de conteúdo `<title>` dentro da `<head>`, permitindo definir o título do documento, que poder ser visto na *barra de título* ou *aba* da janela do navegador. Caso contrário, a página não será um documento HTML válido.

Outra configuração muito importante, principalmente em documentos HTML cujo conteúdo é escrito em um idioma como o português, que contém caracteres "especiais" (acentos e cedilha), é a codificação/conjunto de caracteres, chamada de **encoding** ou **charset**.

Podemos configurar qual codificação queremos utilizar em nosso documento por meio da configuração de `charset` na tag `<meta>`. Um dos valores mais comuns usados hoje em dia é o **UTF-8**, também chamado de **Unicode**. Há outras possibilidades, como o **latin1**, muito usado antigamente.

O **UTF-8** é a recomendação atual para encoding na Web por ser amplamente suportada em navegadores e editores de código, além de ser compatível com praticamente todos os idiomas do mundo. É o que usaremos no curso.

```
<html>
    <head>
      <meta charset="utf-8">
      <title>MusicDot</title>
    </head>
    <body>

    </body>
</html>
```

5.3 A TAG <BODY>

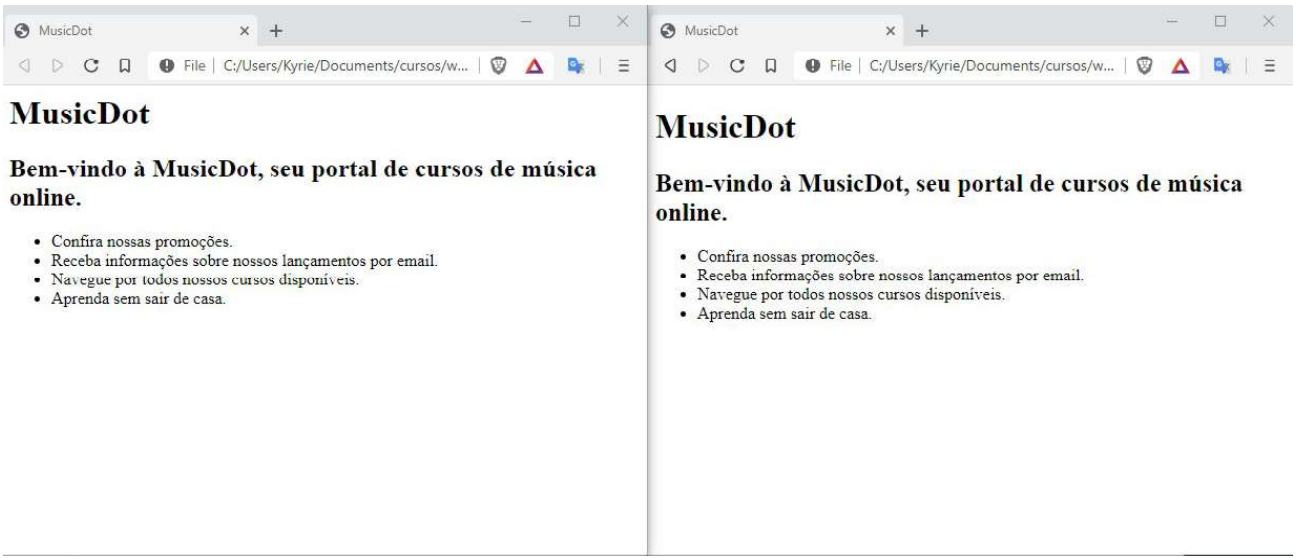
A tag `<body>` contém o corpo de um documento HTML, que é exibido pelo navegador em sua janela, ou seja, todo o conteúdo visível do site. É necessário que o `<body>` tenha ao menos um elemento "filho", ou seja, uma ou mais tags HTML dentro dele.

```
<html>
  <head>
    <meta charset="utf-8">
    <title>MusicDot</title>
  </head>
  <body>
    <h1>A MusicDot</h1>
  </body>
</html>
```

Nesse exemplo, usamos a tag `<h1>`, que indica o título principal da página.

5.4 A INSTRUÇÃO DOCTYPE

O `DOCTYPE` não é uma tag HTML, mas uma instrução especial. Ela indica para o navegador qual **versão do HTML** deve ser utilizada para exibir a página. Quando não colocamos essa instrução a página é exibida numa espécie de "*modo de compatibilidade*" na qual algumas tags e estilizações não funcionam 100% corretamente. Principalmente as tags e estilizações mais atuais (lançadas na versão 5 do HTML). Inclusive é possível ver a diferença na folha de estilos padrão que o navegador usa quando não colocamos essa instrução.



A imagem da esquerda é a página **sem** `Doctype` e a imagem da direita é a página **com** `Doctype`. Dá para ver que existe uma leve diferença entre as duas páginas, principalmente com relação aos espaçamentos.

Utilizaremos `<!DOCTYPE html>`, que indica para o navegador a utilização da versão mais recente do HTML - a versão 5, atualmente*.

Há muitas possibilidades mais complicadas nessa parte de `DOCTYPE` que eram usados em versões anteriores do HTML e do XHTML. Hoje em dia, nada disso é mais importante. O recomendado é **sempre usar a última versão do HTML**, usando a declaração de `DOCTYPE` simples:

```
<!DOCTYPE html>
```

A declaração do DOCTYPE, pode ser escrita toda em maiúsculo ou toda em minúsculo ou com a primeira letra maiúscula: `<!DOCTYPE HTML>`, `<!DOCTYPE html>`, `<!Doctype HTML>`, `<!Doctype html>`, `<!doctype html>`, `<!doctype HTML>`. O resultado será o mesmo para todos os casos.

**Obs: desde maio de 2019 o desenvolvimento do HTML é mantido pelo W3C (World Wide Web Consortium) https://www.w3.org/, WHATWG e comunidade de desenvolvedores, e sua especificação é aberta no Github https://github.com/whatwg/html, e desde este movimento o HTML é considerado um "padrão vivo" (living standard) onde sua versão a partir da 5 é atualizada continuamente.*

Capítulo 7

ESTILIZANDO COM CSS

Quando escrevemos o HTML, marcamos o conteúdo da página com tags que melhor representam o significado daquele conteúdo. Quando abrimos a página no navegador é possível perceber que ele mostra as informações com estilos diferentes.

Um h1, por exemplo, por padrão é apresentado em negrito numa fonte maior. Parágrafos de texto são espaçados entre si, e assim por diante. Isso quer dizer que o navegador tem um *estilo padrão* para as tags que usamos. Porém para fazer sites bonitos, ou com o visual próximo de uma dada identidade visual (design), vamos precisar *personalizar a apresentação padrão dos elementos* da página.

Antigamente, isso era feito no próprio HTML. Caso houvesse a necessidade de um título ser vermelho, era só fazer:

```
<h1><font color="red">MusicDot anos 90</font></h1>
```

Além da tag `<font>` , várias outras tags de estilo existiam. Mas isso é passado. Hoje em dia **tags HTML para estilo são má prática** e jamais devem ser usadas, são interpretadas apenas para o modo de compatibilidade.

Em seu lugar, surgiu o **CSS** (*Cascading Style Sheet* ou folha de estilos em cascata), que é uma outra linguagem, separada do HTML, com objetivo único de cuidar da estilização da página. A vantagem é que o CSS é bem mais robusto que o HTML para estilização, como veremos. Mas, principalmente, escrever formatação visual misturado com conteúdo de texto no HTML se mostrou algo impraticável. O CSS resolve isso separando as coisas; regras de estilo não aparecem mais no HTML, apenas no CSS.

7.1 SINTAXE E INCLUSÃO DE CSS

A sintaxe do CSS tem estrutura simples: é uma declaração de propriedades e valores separados por um sinal de dois pontos ":", e cada propriedade é separada por um sinal de ponto e vírgula ";" da seguinte maneira:

```
color: blue;
background-color: yellow;
```

O elemento que receber essas propriedades será exibido com o texto na cor azul e com o fundo amarelo. Essas propriedades podem ser declaradas de três maneiras diferentes.

Atributo style

A primeira delas é com o atributo `style` no próprio elemento:

```
<p style="color: blue; background-color: yellow;">
O conteúdo desta tag será exibido em azul com fundo amarelo no navegador!
</p>
```

Mas tínhamos acabado de discutir que uma das grandes vantagens do CSS era manter as regras de estilo fora do HTML. Usando esse atributo `style` não parece que fizemos isso. Justamente por isso não se recomenda esse tipo de uso na prática, mas sim os que veremos a seguir.

A tag style

A outra maneira de se utilizar o CSS é declarando suas propriedades dentro de uma tag `<style>`.

Como estamos declarando as propriedades visuais de um elemento em outro lugar do nosso documento, precisamos indicar de alguma maneira a qual elemento nos referimos. Fazemos isso utilizando um **seletor CSS**. É basicamente uma forma de buscar certos elementos dentro da página que receberão as regras visuais que queremos.

No exemplo a seguir, usaremos o seletor que pega todas as tags `p` e altera sua cor e background:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sobre a MusicDot</title>
    <style>
      p {
        color: blue;
        background-color: yellow;
      }
    </style>
  </head>
  <body>
    <p>
      O conteúdo desta tag será exibido em azul com fundo amarelo!
    </p>
    <p>
      <strong>Também</strong> será exibido em azul com fundo amarelo!
    </p>
  </body>
</html>
```

O código dentro da tag `<style>` indica que estamos alterando a cor e o fundo de todos os elementos com tag `p`. Dizemos que selecionamos esses elementos pelo nome de sua tag, e aplicamos certas propriedades CSS apenas neles.

Revisando então a estrutura de uso do CSS:

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

```
}
```

Algumas propriedades contém "subpropriedades" que modificam uma parte específica daquela propriedade que vamos trabalhar, sendo sua sintaxe:

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

No exemplo abaixo, em ambos os casos, trabalhamos com a propriedade `text`, que estiliza a aparência do texto do seletor informado. Podemos especificar quais propriedades específicas do texto queremos modificar, no caso `text-align` o alinhamento do texto, e com `text-decoration` colocamos o efeito de sublinhado.

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

Arquivo externo

A terceira maneira de declararmos os estilos do nosso documento é com um arquivo externo com a extensão `.css`. Para que seja possível declarar nosso CSS em um arquivo à parte, precisamos indicar em nosso documento HTML uma ligação entre ele e a folha de estilo (arquivo com a extensão `.css`).

Além da melhor organização do projeto, a folha de estilo externa traz ainda as vantagens de manter nosso HTML mais limpo e do reaproveitamento de uma mesma folha de estilos para diversos documentos.

A indicação de uso de uma folha de estilos externa deve ser feita dentro da tag `<head>` de um documento HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>MusicDot | Sobre a empresa</title>
    <!-- Inclusão do arquivo CSS -->
    <link rel="stylesheet" href="estilos.css">
  </head>
  <body>
    <p>
      O conteúdo desta tag será exibido em azul com fundo amarelo!
    </p>
    <p>
      <strong>Também</strong> será exibido em azul com fundo amarelo!
    </p>
  </body>
</html>
```

E dentro do arquivo `estilos.css` colocamos apenas o conteúdo do CSS:

```
p {
```

```
  color: blue;
  background-color: yellow;
}
```


7.2 PROPRIEDADES TIPOGRÁFICAS E FONTES

Da mesma maneira que alteramos cores, podemos alterar o texto. Podemos definir fontes com o uso da propriedade `font-family` .

A propriedade `font-family` pode receber seu valor com ou sem aspas dependendo da sua composição, por exemplo, quando uma fonte tem o nome separado por *espaço*.

Por padrão, os navegadores mais conhecidos exibem texto em um tipo que conhecemos como "serif". As fontes mais conhecidas (e comumente utilizadas como padrão) são "Times" e "Times New Roman", dependendo do sistema operacional. Elas são chamadas de **fontes serifadas** pelos pequenos ornamentos em suas terminações.

Podemos alterar a família de fontes que queremos utilizar em nosso documento para a família "sans-serif" (sem serifas), que contém, por exemplo, as fontes "Arial" e "Helvetica". Podemos também declarar que queremos utilizar uma família de fontes "monospace" como, por exemplo, a fonte "Courier".

Obs: Fontes monospace podem ser tanto com serifa ou sem serifa. Monospace quer dizer apenas que todas as letras possuem o mesmo tamanho

```
h1 {
  font-family: serif;
}

h2 {
  font-family: sans-serif;
}
```

```
p {
  font-family: monospace;
}
```

É possível, e muito comum, declararmos o nome de algumas fontes que gostaríamos de verificar se existem no computador, permitindo que tenhamos um controle melhor da forma como nosso texto será exibido.

Em nosso projeto, as fontes não têm ornamentos, vamos declarar essa propriedade para todo o documento por meio do seu elemento `body`:

```
body {
  font-family: "Helvetica", "Lucida Grande", sans-serif;
}
```

Nesse caso, o navegador verificará se a fonte "Helvetica" está disponível e a utilizará para exibir os textos de todos os elementos do nosso documento que, por cascata, herdarão essa propriedade do elemento `body`.

Caso a fonte "Helvetica" não esteja disponível, o navegador verificará a disponibilidade da próxima fonte declarada, no nosso exemplo a "Lucida Grande". Caso o navegador não encontre também essa fonte, ele solicita qualquer fonte que pertença à família "sans-serif", declarada logo a seguir, e a utiliza para exibir o texto, não importa qual seja ela.

Temos outras propriedades para manipular a fonte, como a propriedade `font-style`, que define o estilo da fonte que pode ser: `normal` (normal na vertical), `italic` (inclinada) e `oblique` (oblíqua).

7.3 ALINHAMENTO E DECORAÇÃO DE TEXTO

Já vimos uma série de propriedades e subpropriedades que determinam o tipo e estilo da fonte. Vamos conhecer algumas maneiras de alterarmos as disposições dos textos.

No exemplo a seguir vamos mudar o alinhamento do texto com a propriedade `text-align`.

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

O exemplo determina que todos os parágrafos da nossa página tenham o texto alinhado para a direita. Também é possível determinar que um elemento tenha seu conteúdo alinhado ao centro ao definirmos o valor `center` para a propriedade `text-align`, ou então definir que o texto deve ocupar toda a largura do elemento aumentando o espaçamento entre as palavras com o valor `justify`. Por padrão o texto é alinhado à esquerda, com o valor `left`, porém é importante lembrar que essa propriedade propaga-se em cascata.

É possível configurar também uma série de espaçamentos de texto com o CSS:

```
p {
  line-height: 3px; /* tamanho da altura de cada linha */
  letter-spacing: 3px; /* tamanho do espaço entre cada letra */
  word-spacing: 5px; /* tamanho do espaço entre cada palavra */
  text-indent: 30px; /* tamanho da margem da primeira linha do texto */
}
```

7.4 IMAGEM DE FUNDO

A propriedade `background-image` permite indicar um arquivo de imagem para ser exibido ao fundo do elemento. Por exemplo:

```
h1 {
  background-image: url(sobre-background.jpg);
}
```

Com essa declaração, o navegador vai requisitar um arquivo `sobre-background.jpg` , que deve estar na mesma pasta do arquivo CSS onde consta essa declaração. Mas podemos também passar um endereço da web para pegar imagens remotamente:

```
body {
  background-image: url(https://i.imgur.com/uAhjMNd.jpg);
}
```


7.5 BORDAS

As propriedades do CSS para definirmos as **bordas** de um elemento nos apresentam uma série de opções. Podemos, para cada borda de um elemento, determinar sua cor, seu estilo de exibição e sua largura. Por exemplo:

```
body {
  border-color: red;
  border-style: solid;
  border-width: 1px;
}
```

A propriedade `border` tem uma forma resumida para escrever os mesmos estilos que adicionamos acima, mas de uma maneira mais simples:

```
body {
  border: 1px solid red;
}
```

Para que o efeito da cor sobre a borda surta efeito, é necessário que a propriedade `border-style` tenha qualquer valor diferente do padrão `none` .

Podemos também falar em qual dos lados do nosso elemento queremos a borda usando a subpropriedade que indica lado:

```
h1 {
  border-top: 1px solid red; /* borda vermelha em cima */
  border-right: 1px solid red; /* borda vermelha à direita */
  border-bottom: 1px solid red; /* borda vermelha embaixo */
  border-left: 1px solid red; /* borda vermelha à esquerda */
}
```

Conseguimos fazer também comentários no CSS usando a seguinte sintaxe:

```
/* deixando o fundo amarelo ouro */
body {
    background-color: gold;
}
```

7.6 CORES NA WEB

Propriedades como `background-color` , `color` , `border-color` , entre outras aceitam uma cor como valor. Existem várias maneiras de definir cores quando utilizamos o CSS.

A primeira, mais simples, é usando o nome da cor:

```
h1 {
    color: red;
}

h2 {
    background-color: yellow;
}
```

O difícil é acertar a exata variação de cor que queremos no design e também cada navegador tem o seu padrão de cor para os nomes de cores. A W3C obriga que todos os navegadores tenham pelo menos 140 nomes de cores padronizados, mas existem mais de 16 milhões de cores na web e seria extremamente complicado nomear cada uma delas. Por isso, é bem incomum usarmos cores com seus nomes. O mais comum é definir a cor com base em sua composição RGB.

RGB é o sistema de cor usado nos monitores, já que cada pixel nos monitores possuem 3 leds (um

vermelho, um verde e um azul) e a combinação dessas 3 cores formam todas as outras 16 milhões de cores que vemos nos monitores. Podemos escolher a intensidade de cada um desses três leds básicos, numa escala de 0 a 255.

Um amarelo forte, por exemplo, tem 255 de Red, 255 de Green e 0 de Blue (255, 255, 0). Se quiser um laranja, basta diminuir um pouco o verde (255, 200, 0). E assim por diante.

No CSS, podemos escrever as cores tendo como base sua composição RGB. Aliás, no CSS3 - que veremos melhor depois - há até uma sintaxe bem simples pra isso:

```
h3 {
    color: rgb(255, 200, 0);
}
```

Essa sintaxe funciona nos browsers mais modernos e até alguns browsers super antigos mas não é a mais comum na prática, por questões de compatibilidade. O mais comum é a **notação hexadecimal**. Essa sintaxe tem suporte universal nos navegadores e é mais curta de escrever, apesar de ser mais enigmática.

```
h3 {
    background-color: #f2eded;
}
```

No fundo, porém, as duas formas são baseadas no sistema RGB. Na notação hexadecimal (que começa com #), temos 6 caracteres, os primeiros 2 indicam o canal Red, os dois seguintes, o Green, e os dois últimos, Blue; ou seja, RGB. E usamos a matemática pra escrever menos, trocando a base numérica de decimal para hexadecimal.

Na base hexadecimal, os algarismos vão de zero a quinze (ao invés do zero a nove da base decimal comum). Para representar os algarismos de dez a quinze, usamos letras de A a F. Nessa sintaxe, portanto, podemos utilizar números de 0-9 e letras de A-F.

Há uma conta por trás dessas conversões, mas seu editor de imagens deve ser capaz de fornecer ambos os valores para você sem problemas. Um valor 255 vira FF na notação hexadecimal. A cor **#f2eded**, por exemplo, é equivalente a **rgb(242, 237, 237)**, um cinza claro.

Vale aqui uma dica quanto ao uso de cores hexadecimais, toda vez que os caracteres presentes na composição da base se repetirem, estes podem ser simplificados. Então um número em hexadecimal **3366ff**, pode ser simplificado para **36f**.

Obs: os 3 pares de números devem ser iguais entre si, ou seja, se tivermos um hexadecimal #33aabc não podemos simplificar nada do código.